ΜΕΤΑΦΡΑΣΤΕΣ

ΑΘΑΝΑΣΙΟΣ ΡΟΥΔΗΣ 5098
ΣΤΥΛΙΑΝΟΣ ΣΗΜΑΝΤΗΡΑΚΗΣ 5127

ΠΕΡΙΕΧΟΜΕΝΑ

	Αριθμός Σελίδας
1)Τρόπος εκτέλεσης	2
2)Αρχεία για έλεγχο ορθής λειτουργίας	2
3)Λεκτικός αναλυτής	3
4)Συντακτικός αναλυτής	7
5)Ενδιάμεσος κώδικας	37
6)Πίνακας συμβόλων	38
7)Τελικός κώδικας	42

1)Τρόπος εκτέλεσης

Το αρχείο του μεταγλωττιστή είναι το : cpy.py .Για να εκτελέσετε τον μεταγλωττιστή πρέπει να ανοίξετε ένα τερματικό στο φάκελο στον οποίο έχετε αποθηκεύσει το cpy.py και να εκτελέσετε την ακόλουθη εντολή:

\the_path_of_the_file>/python cpy.py nameOfTheFileToTest.cpy

Ο κώδικας που χρησιμοποιήθηκε για να είναι εφικτή αυτή η εντολή φαίνεται στην Εικόνα 1.

```
1490    #Gia to diabasma apo ta arguments
1491    parser = argparse.ArgumentParser(description='Compile a file.')
1492    parser.add_argument('input_file', help='Path to the input file')
1493
1494    args = parser.parse_args()
1495
1496    fileToCompile = args.input_file
1497
1498    try:
1499        f = open(fileToCompile)
1500    except Exception as e:
1501        print(e)
```

Εικόνα 1

Ο ενδιάμεσος κώδικας παράγεται στο αρχείο intermediate-for-(nameOfTheFileToTest.cpy).int και ο πίνακας συμβόλων παράγεται στο αρχείο symbol-table-for-(nameOfTheFileToTest.cpy).sym .Ο τελικός κώδικας παράγεται στο αρχείο assembly-for-(nameOfTheFileToTest.cpy).asm.

2)Αρχεία για έλεγχο ορθής λειτουργίας

Τα αρχεία για το έλεγχο ορθής λειτουργίας είναι:

i) test.cpy , το δοθέν αρχείο για έλεγχο.

ii) onlyMainTest.cpy , περιέχει ένα πρόγραμμα μόνο με την συνάρτηση main.

iii) moreThanOneDeclarationsTest.cpy, περιέχει ένα πρόγραμμα με παραπάνω από μια δηλώσεις.

iv) limitsTest.cpy , περιέχει ένα πρόγραμμα με οριακές τιμές.

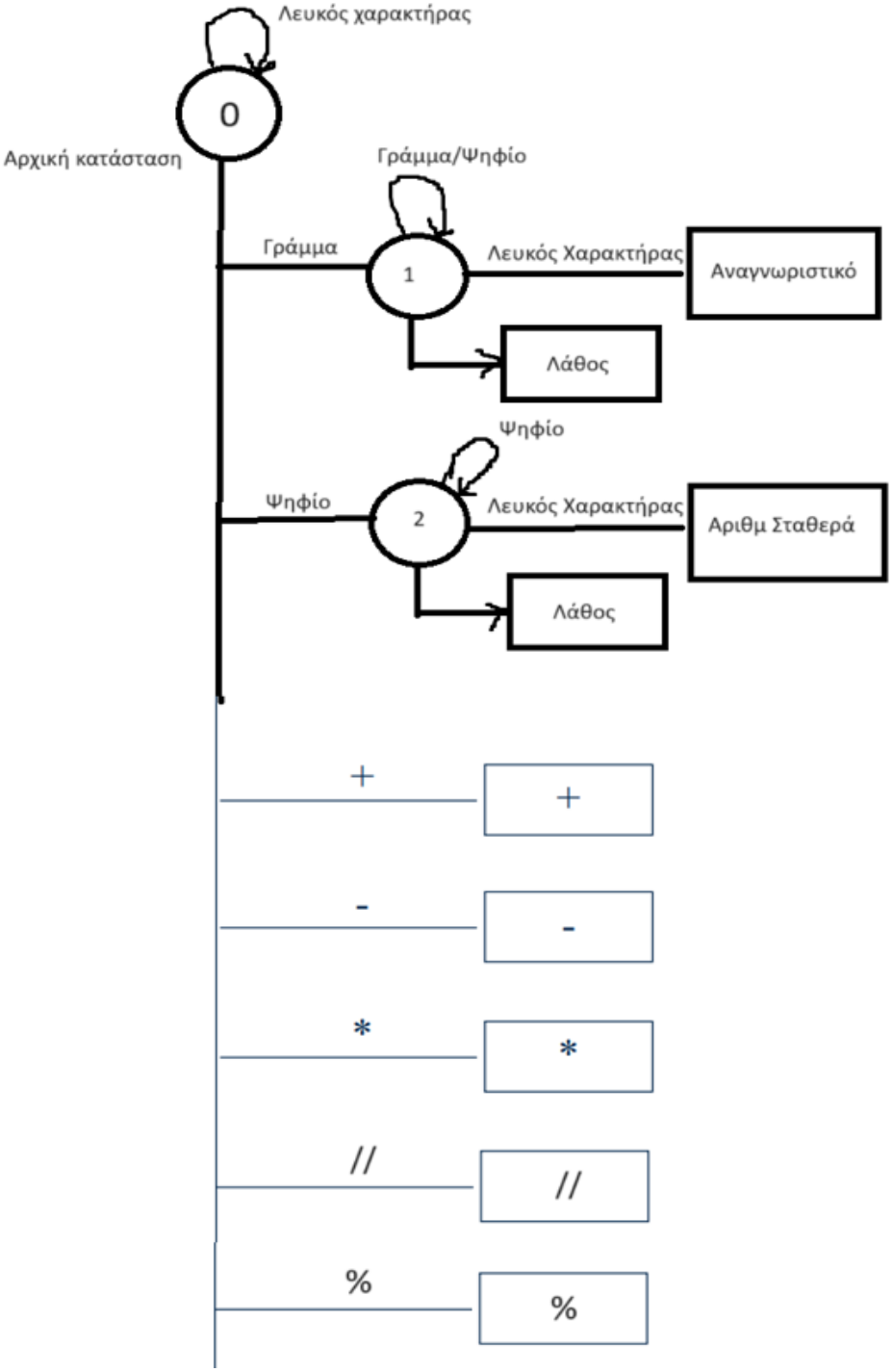
v)smallTest.cpy , περιέχει ένα πρόγραμμα από τις διαφάνειες .

vi)ifWhileTest.cpy , περιέχει ένα πρόγραμμα από τις διαφάνειες .

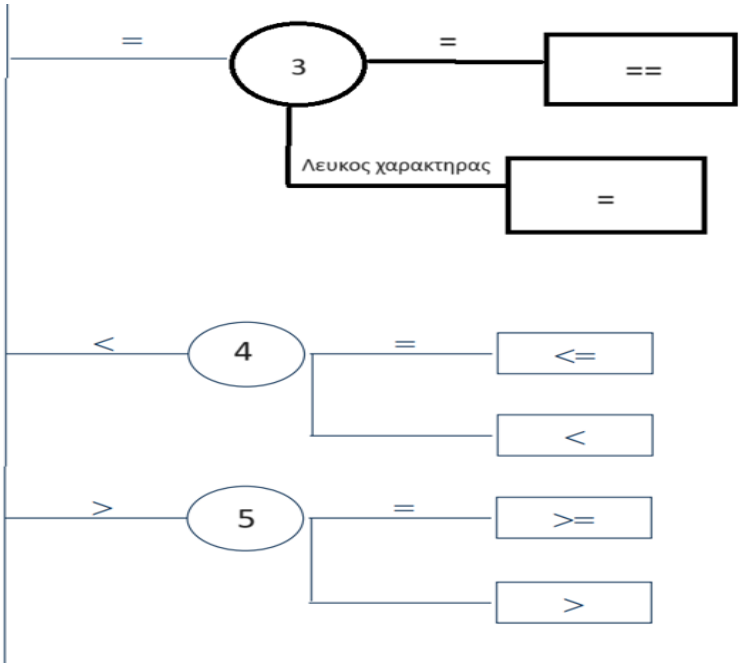
vii) finalCodeExampleTest.cpy , περιέχει ένα πρόγραμμα από τις διαφάνειες .

3)Λεκτικός αναλυτής

Για την δημιουργία του λεκτικού αναλυτή χρησιμοποιήθηκε το αυτόματο των Εικόνων 2,3,4 & 5 . Πιο συγκεκριμένα ο λεκτικός αναλυτής διαβάζει έναν έναν τους χαρακτήρες από το αρχείο εισόδου και αποφασίζει σε ποια κατάσταση του αυτόματου βρίσκεται.

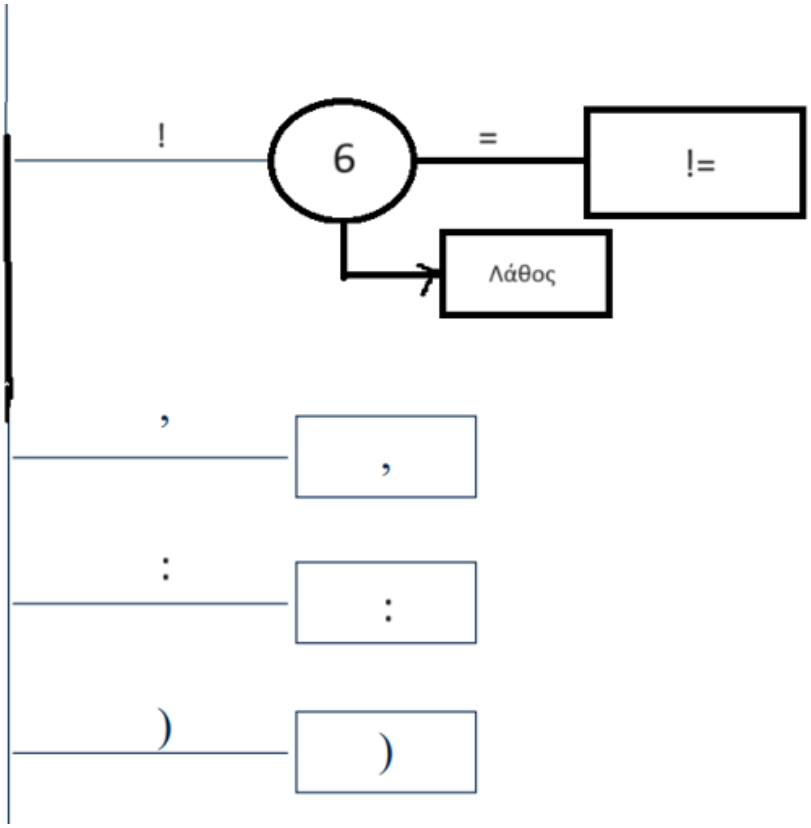


Εικόνα 2

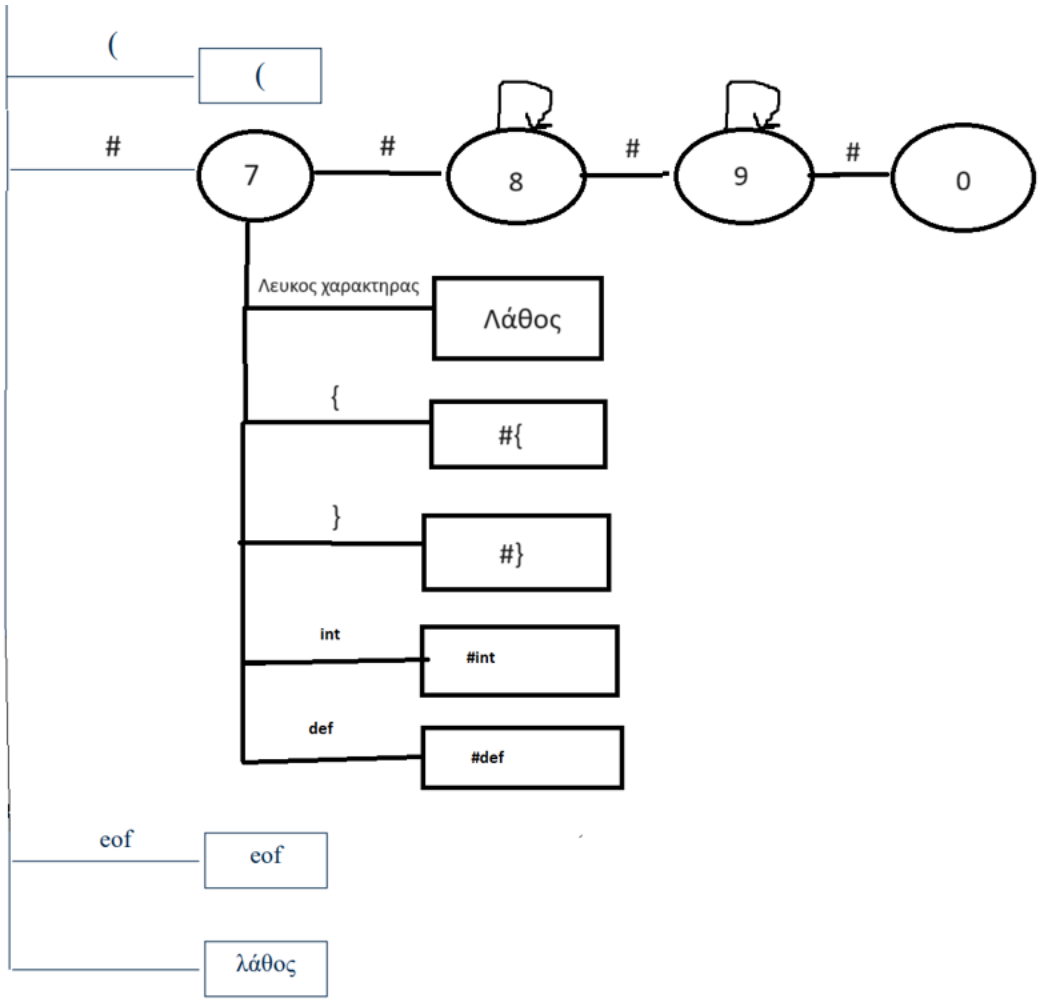


Εικόνα 3

Επίσης σε αυτό το στάδιο γίνεται ο έλεγχος για τις οριακές τιμές που δίνονται στην εκφώνηση. Τέλος όταν εντοπιστεί κάποιο λάθος είτε από το αυτόματο είτε από τις οριακές τιμές εκτυπώνεται το αντίστοιχο μήνυμα λάθους και μεταγλώττιση τερματίζεται.



Εικόνα 4



Εικόνα 5

Εξήγηση αυτομάτου/κώδικα:

Καταστάσεις

- **Κατάσταση 0**: Αρχική κατάσταση. Διαβάζει τον τρέχοντα χαρακτήρα και αποφασίζει την επόμενη κατάσταση με βάση τον χαρακτήρα αυτό.
    - Αν ο χαρακτήρας είναι κενό, newline, carriage return, tab ή vertical tab, παραμένει στην κατάσταση 0.
    - Αν ο χαρακτήρας είναι γράμμα, μεταβαίνει στην κατάσταση 1.
    - Αν ο χαρακτήρας είναι ψηφίο, μεταβαίνει στην κατάσταση 2.
    - Αν ο χαρακτήρας είναι =, μεταβαίνει στην κατάσταση 3.
    - Αν ο χαρακτήρας είναι <, μεταβαίνει στην κατάσταση 4.
    - Αν ο χαρακτήρας είναι >, μεταβαίνει στην κατάσταση 5.
    - Αν ο χαρακτήρας είναι !, μεταβαίνει στην κατάσταση 6.

    - Αν ο χαρακτήρας είναι `#`, μεταβαίνει στην κατάσταση 7.
    - Αν ο χαρακτήρας είναι `+`, επιστρέφει το token `["addtoken", line]`.
    - Αν ο χαρακτήρας είναι `-`, επιστρέφει το token `["subtoken", line]`.
    - Αν ο χαρακτήρας είναι `*`, επιστρέφει το token `["multoken", line]`.
    - Αν ο χαρακτήρας είναι `/`, μεταβαίνει στην κατάσταση 14.
    - Αν ο χαρακτήρας είναι `%`, επιστρέφει το token `["modtoken", line]`.
    - Αν ο χαρακτήρας είναι `,`, επιστρέφει το token `["commatoken", line]`.
    - Αν ο χαρακτήρας είναι `:`, επιστρέφει το token `["anwkatwtoken", line]`.
    - Αν ο χαρακτήρας είναι `(`, επιστρέφει το token `["leftpartoken", line]`.
    - Αν ο χαρακτήρας είναι `)`, επιστρέφει το token `["rightpartoken", line]`.
    - Για οποιονδήποτε άλλο χαρακτήρα, εμφανίζει μήνυμα λάθους και σταματά την εκτέλεση.
- **Κατάσταση 1**: Συλλέγει λέξεις. Αν ο χαρακτήρας είναι γράμμα ή ψηφίο, τον προσθέτει στη λέξη. Διαφορετικά, επιστρέφει το κατάλληλο token.
    - Αν η λέξη υπάρχει στη `commandList`, επιστρέφει το token `["commandtoken", line, word]`.
    - Αν η λέξη είναι μοναδικό αναγνωριστικό (έως 30 χαρακτήρες), επιστρέφει το token `["anagnoristikotoken", line, word]`.
- **Κατάσταση 2**: Συλλέγει αριθμούς. Αν ο χαρακτήρας είναι ψηφίο, τον προσθέτει στον αριθμό. Διαφορετικά, επιστρέφει το token `["numbertoken", line, int(number)]` αν ο αριθμός είναι εντός ορίων.
- **Κατάσταση 3**: Ελέγχει αν ο επόμενος χαρακτήρας είναι `=`.
    - Αν είναι, επιστρέφει το token `["isothtatoken", line]`.
    - Διαφορετικά, επιστρέφει το token `["ana8eshtoken", line]`.
- **Κατάσταση 4**: Ελέγχει αν ο επόμενος χαρακτήρας είναι `=`.
    - Αν είναι, επιστρέφει το token `["mikisotoken", line]`.
    - Διαφορετικά, επιστρέφει το token `["mikroterotoken", line]`.
- **Κατάσταση 5**: Ελέγχει αν ο επόμενος χαρακτήρας είναι `=`.
    - Αν είναι, επιστρέφει το token `["megisotoken", line]`.
    - Διαφορετικά, επιστρέφει το token `["megaluterotoken", line]`.
- **Κατάσταση 6**: Ελέγχει αν ο επόμενος χαρακτήρας είναι `=`.
    - Αν είναι, επιστρέφει το token `["diaforotoken", line]`.
    - Διαφορετικά, εμφανίζει μήνυμα λάθους και σταματά την εκτέλεση.
- **Κατάσταση 7**: Ελέγχει για σχόλια και σύμβολα `{`, `}` ή αναγνωρίζει ειδικά tokens `#i` (για `int`) και `#d` (για `def`).
- **Κατάσταση 8**: Διαχειρίζεται τα σχόλια.
- **Κατάσταση 9-12**: Ελέγχουν για τις ειδικές λέξεις `#int` και `#def`.
- **Κατάσταση 13**: Συνεχίζει τα σχόλια.
- **Κατάσταση 14**: Ελέγχει για το `//` (ακέραια διαίρεση).

Αν η συνάρτηση φτάσει στο τέλος του αρχείου, επιστρέφει το token `["EOFtoken", line]`.

4)Συντακτικός αναλυτής

Για την υλοποίηση του συντακτικού αναλυτή χρησιμοποιήθηκε η γραμματική της γλώσσας cpy . Ακολουθεί η γραμματική της cpy η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
startRule
        :
          declarations
          def_function
      call_main_part
        ;

def_function
        : 'def' ID '(' id_list ')' ':'
            '#{'
                  Declarations
                  ( def_function )*
                  globals
               statements

            '#}'
        ;

declarations
        : ( declaration_line )*
        ;

declaration_line
        : '#int' id_list
        ;

globals
       : ( globals_line )*
        ;

globals_line
        : 'global' id_list
;

statement
        : simple_statement
        | structured_statement
        ;
```

```
statements
        : statement+
        ;

simple_statement
        : assignment_stat
        | print_stat
        | return_stat
        ;

structured_statement
        : if_stat
        | while_stat
        ;


assignment_stat
        : ID '='
            ( expression
                        | 'int' '(' 'input' '(' ')' ')'
            )
        ;

print_stat
        : 'print' '(' expression ')'
        ;

return_stat
        : 'return' expression
        ;

if_stat

       :  'if' condition':'
               ( statement
               )
          ( 'elif' condition ':'
               ( statement
               )
          )
          ( 'else'':'
               ( statement
       ;       )
          )?
while_stat
```

```
        :  'while' condition ':'
           '#{'
                ( statement
                )
        ;  '#}'

id_list
        :    ID ( ',' ID )*
        |
        ;

expression
        :  optional_sign term
           ( ADD_OP term )*
        ;

term
        :  factor
           ( MUL_OP factor )*
        ;



Factor
        :       INTEGER
        |       '(' expression ')'
        |       ID idtail
        ;


idtail
         : '('actual_par_list ')'
         |
         ;

actual_par_list
         :      expression ( ',' expression )*
         |
         ;


Optional_sign
         :    ADD_OP
         |
         ;

condition
         : bool_term ( 'or' bool_term )*
         ;
```

```
bool_term
        : bool_factor ( 'and' bool_factor )*
        ;

bool_factor
        : 'not' '[' condition ']'
        | '[' condition ']'
        | expression REL_OP expression
        ;

call_main_part :
        '#def main'
        declarations
        statements
        ;
```

Ο συντακτικός αναλυτής διαβάζει σε κάθε περίπτωση τον λεκτικό αναλυτή έναν πίνακα ο οποίος περιέχει το token, τη γραμμή και σε κάποιες περιπτώσεις οτιδήποτε άλλο είναι χρήσιμο . Όταν εντοπιστεί κάποιο λάθος εκτυπώνεται το αντίστοιχο μήνυμα λάθους και μεταγλώττιση τερματίζεται.

Υλοποίηση γραμματικής σε python:

Κανόνας if_stat:

- • **Έλεγχος αν το token είναι `commandtoken` και αν η εντολή είναι `if` ή `elif`:**
    - Αν ναι, συνεχίζει με την επεξεργασία.
    - Διαφορετικά, η συνάρτηση επιστρέφει.
- **Έλεγχος και Ανάγνωση Συνθήκης:**
    - Καλεί τη συνάρτηση `isCondition(token)` για να ελέγξει αν το επόμενο token αποτελεί συνθήκη.
    - Αν η συνθήκη είναι έγκυρη, αποθηκεύει τις λίστες `conditionTrue` και `conditionFalse`.
    - Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.
- **Backpatch και Ανάγνωση Επόμενου Token:**
    - Ενημερώνει τις λίστες `conditionTrue` με την επόμενη τετράδα.
    - Διαβάζει το επόμενο token.
- **Έλεγχος για : Μετά το `if` ή `elif`:**
    - Αν το token είναι `anwkatwtoken` (δηλαδή :), συνεχίζει.

- Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

- **Έλεγχος και Εκτέλεση Δήλωσης**:

  - Αν η δήλωση είναι έγκυρη (`isStatement(token)`), συνεχίζει.
  - Δημιουργεί μια λίστα `ifList` με την επόμενη τετράδα και προσθέτει μια τετράδα `jump`.
  - Ενημερώνει τη λίστα `conditionFalse` με την επόμενη τετράδα.
  - Αποθηκεύει τη θέση του αρχείου και διαβάζει το επόμενο token.

- **Έλεγχος για Επόμενη Εντολή**:

  - Αν η επόμενη εντολή είναι `elif`, καλεί αναδρομικά την `ifState(token)`.
  - Αν η επόμενη εντολή είναι `else`, καλεί την `elseState(token)`.
  - Αν είναι οποιαδήποτε άλλη εντολή, ενημερώνει τη λίστα `ifList` και επαναφέρει τη θέση του αρχείου.

```
404     def ifState(token):
406         if token[0] == "commandtoken":
407             if token[2] == "if" or token[2] == "elif":
408                 token = lex()
409                 cond = isCondition(token)
410                 if cond[0]:
411                     conditionTrue = cond[1]
412                     conditionFalse = cond[2]
413                     backpatch(conditionTrue, nextQuad())
414                     token = lex()
415                     if token[0] == "anwkatwtoken":
416                         token = lex()
417                         if isStatement(token):
418                             ifList = makeList(nextQuad())
419                             quadList.append(genQuad("jump","_","_","_"))
420                             backpatch(conditionFalse, nextQuad())
421                             seekIndex = f.tell()
422                             token = lex()
423                             if token[0] == "commandtoken":
424                                 if token[2] == "elif":
425                                     x = ifState(token)
426                                     backpatch(ifList,nextQuad())
427                                     return x
428                                 elif token[2] == "else":
429                                     x = elseState(token)
430                                     backpatch(ifList,nextQuad())
431                                     return x
432                                 else:
433                                     backpatch(ifList,nextQuad())
434                                     f.seek(seekIndex)
435                                     return True
436                             else:
437                                 backpatch(ifList,nextQuad())
438                                 f.seek(seekIndex)
439                                 return True
440                         else:
441                             print ("Inside the 'if' at line:", token[1], "a statement was expected")
442                             print("Compilation failed")
443                             exit()
444                     else:
445                         print ("After the 'if' at line:", token[1], "a ':' was expected")
446                         print("Compilation failed")
447                         exit()
448                 else:
449                     print ("After the 'if' at line:", token[1], "a condition was expected")
450                     print("Compilation failed")
451                     exit()
```

Εικόνα 6

- **Έλεγχος Αν Η Εντολή Είναι `else`:**
    - Η συνάρτηση ξεκινά με την επαλήθευση ότι το token που έχει περαστεί είναι η εντολή `else`.
    - Αν είναι, διαβάζει το επόμενο token με τη συνάρτηση `lex()`.

- **Έλεγχος Για `:` Μετά Το `else`:**
    - Αν το επόμενο token είναι `anwkatwtoken` (δηλαδή `:`), προχωρά στην επόμενη γραμμή.
    - Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

- **Έλεγχος και Εκτέλεση Δήλωσης:**
    - Διαβάζει το επόμενο token και ελέγχει αν αυτό αποτελεί μια έγκυρη δήλωση χρησιμοποιώντας τη συνάρτηση `isStatement(token)`.
    - Αν η δήλωση είναι έγκυρη, επιστρέφει `True`.
    - Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

- **Λάθη και Εξαιρέσεις:**
    - Αν βρεθεί οποιοδήποτε πρόβλημα (όπως απουσία `:` ή μη έγκυρη δήλωση), εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

```
453     def elseState(token):
454         if token[2] == "else":
455             token = lex()
456             if token[0] == "anwkatwtoken":
457                     token = lex()
458                     if isStatement(token):
459                         return True
460                     else:
461                         print ("Inside the 'else' at line:", token[1], "a statement was expected")
462                         print("Compilation failed")
463                         exit()
464             else:
465                     print ("After the 'else' at line:", token[1], "a ':' was expected")
466                     print("Compilation failed")
467                     exit()
```

Εικόνα 7

Κανόνας condition:

- **Αρχικοποίηση Λιστών:**
    - Οι λίστες `Btrue` και `Bfalse` αρχικοποιούνται ως κενές.

- **Έλεγχος και Ανάγνωση Λογικού Όρου**:
    - Καλεί τη συνάρτηση `isBoolTerm(token)` για να ελέγξει αν το επόμενο token αποτελεί λογικό όρο.
    - Αν ο λογικός όρος είναι έγκυρος, αποθηκεύει τις λίστες `Q1true` και `Q1false`.
    - Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

- **Αρχικοποίηση Λιστών Btrue και Bfalse**:
    - Οι λίστες `Btrue` και `Bfalse` αρχικοποιούνται με τις τιμές των `Q1true` και `Q1false` αντίστοιχα.

- **Επανάληψη για τον Τελεστή `or`**:
    - Αποθηκεύει τη θέση του αρχείου και διαβάζει το επόμενο token.
    - Αν το token είναι η εντολή `or`, συνεχίζει την επεξεργασία.
    - Καλεί τη συνάρτηση `isBoolTerm(token)` για να ελέγξει τον επόμενο λογικό όρο μετά το `or`.
    - Αν ο λογικός όρος δεν είναι έγκυρος, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.
    - Ενημερώνει τις λίστες `Btrue` και `Bfalse` με τις νέες τιμές.

- **Επιστροφή Αποτελέσματος**:
    - Αν βρεθεί έγκυρος λογικός όρος, επιστρέφει μια λίστα με τρεις τιμές: [True, Btrue, Bfalse].
    - Αν όχι, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

```
def isCondition(token):
    Btrue = []
    Bfalse = []
    global seekIndex
    bterm1 = isBoolTerm(token)
    if bterm1[0]:
        Q1true = bterm1[1]
        Q1false = bterm1[2]
        Btrue = Q1true
        Bfalse = Q1false
        seekIndex = f.tell()
        token = lex()
        if token[0] == "commandtoken":
            while token[2]=="or":
                backpatch(Bfalse,nextQuad())
                seekIndex = f.tell()
                token = lex()
                bterm2 = isBoolTerm(token)
                if bterm2[0] == False:
                    print("After the 'or' at line:", token[1], "a boolean term was expected")
                    print("Compilation failed")
                    exit()
                else:
                    Q2true = bterm2[1]
                    Q2false = bterm2[2]
                    Btrue = mergeList(Btrue,Q2true)
                    Bfalse = Q2false
                    f.seek(seekIndex)
            return [True,Btrue,Bfalse]
        else:
            f.seek(seekIndex)
            return [True,Btrue,Bfalse]
    else:
        print("At line:", token[1],"a boolean term was expected")
        print("Compilation failed")
        exit()
```

Εικόνα 8

Κανόνας boolTerm:

- **Αρχικοποίηση Μεταβλητών**:
  - Αρχικοποιεί τις λίστες `Qtrue` και `Qfalse` ως κενές.
- **Έλεγχος και Ανάλυση του Πρώτου Λογικού Παράγοντα**:
  - Καλεί τη συνάρτηση `isBoolFactor(token)` για να ελέγξει τον πρώτο λογικό παράγοντα.
  - Εάν ο πρώτος παράγοντας είναι έγκυρος, αποθηκεύει τις λίστες `R1true` και `R1false`.
  - Εάν δεν είναι έγκυρος, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.
- **Επεξεργασία των Συνθηκών με Τον Τελεστή `and`**:
  - Εάν ο πρώτος παράγοντας είναι έγκυρος, ελέγχει εάν ακολουθεί ο τελεστής `and`.
  - Αν υπάρχει το `and`, επιστρέφει στον επόμενο λογικό παράγοντα.
  - Επαναλαμβάνει αυτήν τη διαδικασία μέχρι να μην υπάρχει πλέον ο τελεστής `and`.
  - Επιστρέφει τις λίστες `Qtrue` και `Qfalse` μετά την επεξεργασία.
- **Επιστροφή των Αποτελεσμάτων**:
  - Επιστρέφει μια λίστα με τρεις τιμές: [True, Qtrue, Qfalse].
  - Αν ο πρώτος λογικός παράγοντας δεν είναι έγκυρος, εμφανίζει μήνυμα λάθους και τερματίζει την εκτέλεση.

```
505
506     def isBoolTerm(token):
507         Qtrue = []
508         Qfalse = []
509         global seekIndex
510         bfactor1 = isBoolFactor(token)
511         if bfactor1[0]:
512             R1true = bfactor1[1]
513             R1false = bfactor1[2]
514             Qtrue = R1true
515             Qfalse = R1false
516             seekIndex = f.tell()
517             token = lex()
518             if token[0] == "commandtoken":
519                 while token[2]=="and":
520                     backpatch(Qtrue,nextQuad())
521                     seekIndex = f.tell()
522                     token = lex()
523                     bfactor2 = isBoolFactor(token)
524                     if bfactor2[0]:
525                         R2true = bfactor2[1]
526                         R2false = bfactor2[2]
527                         Qfalse = mergeList(Qfalse,R2false)
528                         Qtrue = R2true
529                         seekIndex = f.tell()
530                         token = lex()
531                         if token[0]!= "commandtoken":
532                             break
533                     else:
534                         print("After the 'and' at line:", token[1], "a boolean factor was expected")
535                         print("Compilation failed")
536                         exit()
537                 f.seek(seekIndex)
538                 return [True,Qtrue,Qfalse]
539             else:
540                 f.seek(seekIndex)
541                 return [True,Qtrue,Qfalse]
542         else:
543             print("At line:", token[1],"a boolean factor was expected")
544             print("Compilation failed")
545             exit()
```

Εικόνα 9

Κανόνας boolFactor:

- **Αρχικοποίηση Μεταβλητών**:
  - Δημιουργεί τις λίστες `Rtrue` και `Rfalse` κενές.
- **Έλεγχος του Τύπου του Πρώτου token**:
  - Αν το πρώτο token είναι μια εντολή (στοιχείο `commandtoken`), τότε πρέπει να είναι μια έκφραση ή μια συνθήκη.
    - Εάν είναι έκφραση, ελέγχει για τον τελεστή σύγκρισης και μια δεύτερη έκφραση.
    - Εάν είναι συνθήκη με το `not`, αναλύει την αντίστοιχη συνθήκη και επιστρέφει τα αντίθετα σύνολα `Rtrue` και `Rfalse`.
  - Αν το πρώτο token δεν είναι εντολή, τότε μπορεί να είναι είτε μια έκφραση είτε μια συνθήκη.

```
547     def isBoolFactor(token):
549         Rfalse = []
550         if token[0] == "commandtoken":
551             expr1 = isExpression(token)
552             if expr1[0]:
553                 E1place = expr1[1]
554                 token = lex()
555                 if token[0] in relOpList:
556                     if token[0] == "isothtatoken":
557                         relOp = "="
558                     elif token[0] == "mikroterotoken":
559                         relOp = "<"
560                     elif token[0] == "megaluterotoken":
561                         relOp = ">"
562                     elif token[0] == "mikisotoken":
563                         relOp = "<="
564                     elif token[0] == "megisotoken":
565                         relOp = ">="
566                     else:
567                         relOp = "!="
568                     token = lex()
569                     expr2 = isExpression(token)
570                     if expr2[0]:
571                         E2place = expr2[1]
572                         Rtrue = makeList(nextQuad())
573                         quadList.append(genQuad(relOp, E1place,E2place,"_"))
574                         Rfalse = makeList(nextQuad())
575                         quadList.append(genQuad("jump","_","_","_"))
576                         return [True,Rtrue,Rfalse]
577                     else:
578                         print("At line", token[1], "an expression was expected after a relationship opperand")
579                         print("Compilation failed")
580                         exit()
```

Εικόνα 10

- **Εφαρμογή Τελεστή Σύγκρισης**:
    - Εάν το επόμενο token είναι τελεστής σύγκρισης, τότε εκτελεί την αντίστοιχη σύγκριση με την επόμενη έκφραση.
- **Επιστροφή Αποτελεσμάτων**:
    - Επιστρέφει μια λίστα με τρεις τιμές: [True, Rtrue, Rfalse] αν η ανάλυση είναι επιτυχής, αλλιώς επιστρέφει [False].

```
581             elif token[2] == "not":
582                 token = lex()
583                 cond = isCondition(token)
584                 if cond[0]:
585                     Btrue = cond[1]
586                     Bfalse = cond[2]
587                     Rtrue = Bfalse
588                     Rfalse = Btrue
589                     return [True,Rtrue,Rfalse]
590                 else:
591                     print("At line:", token[1],"a boolean factor was expected after 'not")
592                     print("Compilation failed")
593                     exit()
594             else:
595                 return [False]
596         else:
597             expr1 = isExpression(token)
598             if expr1[0]:
599                 E1place = expr1[1]
600                 token = lex()
601                 if token[0] in relOpList:
602                     if token[0] == "isothtatoken":
603                         relOp = "="
604                     elif token[0] == "mikroterotoken":
605                         relOp = "<"
606                     elif token[0] == "megaluterotoken":
607                         relOp = ">"
608                     elif token[0] == "mikisotoken":
609                         relOp = "<="
610                     elif token[0] == "megisotoken":
611                         relOp = ">="
612                     else:
613                         relOp = "!="
614                     token = lex()
615                     expr2 = isExpression(token)
616                     if expr2[0]:
617                         E2place = expr2[1]
618                         Rtrue = makeList(nextQuad())
619                         quadList.append(genQuad(relOp, E1place,E2place,"_"))
620                         Rfalse = makeList(nextQuad())
621                         quadList.append(genQuad("jump","_","_","_"))
622                         return [True,Rtrue,Rfalse]
```

Εικόνα 11

```
621                     quadList.append(genQuad("jump","_","_","_"))
622                     return [True,Rtrue,Rfalse]
623                 else:
624                     print("At line", token[1], "an expression was expected after a relationship opperand")
625                     print("Compilation failed")
626                     exit()
627 
628         else:
629             cond = isCondition(token)
630             if cond[0]:
631                 Btrue = cond[1]
632                 Bfalse = cond[2]
633                 Rtrue = Btrue
634                 Rfalse = Bfalse
635                 return [True,Rtrue,Rfalse]
636             else:
637                 return [False]
```

Εικόνα 12

Κανόνας expression:

- **Αρχικοποίηση Μεταβλητών**:
  - Αρχικά αρχικοποιούνται μεταβλητές όπως η `Eplace`, `seekIndex`, `negSign`.
- **Έλεγχος Προαιρετικού Προσήμου**:
  - Εάν το τρέχον token έχει προαιρετικό πρόσημο (addtoken ή subtoken), αναλύει το επόμενο token.
  - Εάν το πρόσημο είναι subtoken, τότε η μεταβλητή `negSign` ορίζεται ως True.

```
638     def isExpression(token):
639         Eplace = 0
640         global seekIndex
641         global pos
642         global scope
643         seekIndex = f.tell()
644         negSign = False
645         if isOptionalSign(token):
646             if token[0] == "subtoken":
647                 negSign = True
648             token = lex()
649         term1 = isTerm(token)
650         if term1[0]:
651             T1place = term1[1]
652             if negSign:
653                 y = newTemp()
654                 pos=pos+4
655                 newPos=pos
656                 newScope=scope-1
657                 recordStructure.addNewEntity(scopeIndex=newScope,entityName=y,entity_type='TemporaryVariable', offset=newPos)
658                 quadList.append(genQuad("-",0,T1place,y))
659                 T1place = y
660             seekIndex = f.tell()
661             token = lex()
```

Εικόνα 13

- **Ανάλυση Όρων**:
  - Αναλύονται οι αριθμητικοί όροι (terms) της έκφρασης.
  - Αν υπάρχει προαιρετικό πρόσημο "-", εφαρμόζεται αντίστροφος προσανατολισμός.
  - Για κάθε addtoken ή subtoken που ακολουθεί, αναλύεται ένας νέος όρος και πραγματοποιείται η κατάλληλη ανάθεση τιμών.
- **Επιστροφή Αποτελεσμάτων**:
  - Επιστρέφει True και τη θέση του αποτελέσματος (Eplace) αν η ανάλυση είναι επιτυχής.
  - Σε διαφορετική περίπτωση, επιστρέφει False.

```
        while token[0] == "addtoken" or token[0] == "subtoken":
            if token[0] == "addtoken":
                operand = "+"
            else:
                operand = "-"
            token = lex()
            term2 = isTerm(token)
            if term2[0]:
                T2place = term2[1]
                w = newTemp()
                pos=pos+4
                newPos=pos
                newScope=scope-1
                recordStructure.addNewEntity(scopeIndex=newScope,entityName=w,entity_type='TemporaryVariable', offset=newPos)
                quadList.append(genQuad(operand,T1place,T2place,w))
                T1place = w
                seekIndex = f.tell()
                token = lex()
            else:
                print("At line", token[1], "a term was expected after an add or sub opperand. Instead found:", token[0])
                print("Compilation failed")
                exit()
        f.seek(seekIndex)
        Eplace = T1place
        return [True,Eplace]
    else:
        f.seek(seekIndex)
        return [False]
```

Εικόνα 14

Κανόνας optionalSign:

Έλεγχος του token:

- Εάν το τρέχον token είναι addtoken ή subtoken, τότε θεωρείται προαιρετικό πρόσημο και η συνάρτηση επιστρέφει True.
- Σε κάθε άλλη περίπτωση, επιστρέφει False.

```
def isOptionalSign(token):
    global seekIndex
    if token[0] == "addtoken" or token[0] == "subtoken":
        return True
    else:
        return False
```

Εικόνα 15

Κανόνας term:

- **Αρχικοποίηση Μεταβλητών**:
  - Αρχικοποιούνται μεταβλητές όπως η `Tplace`, `seekIndex`.
- **Ανάλυση Παράγοντων**:
  - Αναλύονται οι αριθμητικοί παράγοντες (factors) του όρου.
  - Για κάθε πολλαπλασιασμό ή διαίρεση που ακολουθεί, αναλύεται ένας νέος παράγοντας και πραγματοποιείται η κατάλληλη ανάθεση τιμών.
- **Επιστροφή Αποτελεσμάτων**:
  - Επιστρέφει True και τη θέση του αποτελέσματος (Tplace) αν η ανάλυση είναι επιτυχής.
  - Σε διαφορετική περίπτωση, επιστρέφει False.

```
698     def isTerm(token):
699         Tplace = 0
700         global seekIndex
701         global pos
702         global scope
703         factor1 = isFactor(token)
704         if factor1[0]:
705             F1place = factor1[1]
706             seekIndex = f.tell()
707             token = lex()
708             while token[0] == "multoken" or token[0] == "modtoken" or token[0] == "divtoken":
709                 if token[0] == "multoken":
710                     operand = "*"
711                 elif token[0] == "divtoken":
712                     operand = "/"
713                 else:
714                     operand = "mod"
715                 token = lex()
716                 factor2 = isFactor(token)
717                 if factor2[0]:
718                     F2place = factor2[1]
719                     w = newTemp()
720                     pos=pos+4
721                     newPos=pos
722                     newScope=scope-1
723                     recordStructure.addNewEntity(scopeIndex=newScope,entityName=w,entity_type='TemporaryVariable', offset=newPos)
724                     quadList.append(genQuad(operand,F1place,F2place,w))
725                     F1place = w
726                     seekIndex = f.tell()
727                     token = lex()
728                 else:
729                     print("At line", token[1], "a factor was expected after a multiplication or division opperand. Instead found: ",token[0])
730                     print("Compilation failed")
731                     exit()
732             f.seek(seekIndex)
733             Tplace = F1place
734             return [True,Tplace]
735         else:
736             return [False]
```

Εικόνα 16

Κανόνας factor:

- **Ελέγχος token**:
  - Αν το τρέχον token είναι numbertoken, η συνάρτηση επιστρέφει την τιμή του και True.

- Αν το token είναι αναγνωριστικό (anagnoristikotoken), ελέγχεται εάν ακολουθεί η ουρά του αναγνωριστικού.
- Αν το token είναι αριστερή παρένθεση (leftpartoken), αναλύεται η ακολουθούμενη έκφραση και αναμένεται η αντίστοιχη δεξιά παρένθεση.

- **Επιστροφή Αποτελεσμάτων**:
  - Επιστρέφει True και τη θέση του παράγοντα (Fplace) αν η ανάλυση είναι επιτυχής.
  - Σε διαφορετική περίπτωση, επιστρέφει False.

```
738     def isFactor(token):
739         Fplace = 0
740         global seekIndex
741         global pos
742         global scope
743         if token[0] == "numbertoken":
744             Fplace = token[2]
745             return [True,Fplace]
746         elif token[0] == "anagnoristikotoken":
747             idName = token[2]
748             seekIndex = f.tell()
749             tempSeekIndex = seekIndex
750             token = lex()
751             if idTail(token):
752                 w = newTemp()
753                 pos=pos+4
754                 newPos=pos
755                 newScope=scope-1
756                 recordStructure.addNewEntity(scopeIndex=newScope,entityName=w,entity_type='TemporaryVariable', offset=newPos)
757                 quadList.append(genQuad("par",w,"ret","_"))
758                 quadList.append(genQuad("call",idName,"_","_"))
759                 return [True,w]
760             else:
761                 IDplace = idName
762                 Fplace = IDplace
763                 f.seek(tempSeekIndex)
764                 return [True,Fplace]
765         elif token[0] == "leftpartoken":
766             token = lex()
767             expr = isExpression(token)
768             if expr[0]:
769                 Eplace = expr[1]
770                 Fplace = Eplace
771                 token = lex()
772                 if token[0] == "rightpartoken":
773                     return [True,Fplace]
774                 else:
775                     print("At line", token[1], "a ')' was expected")
776                     print("Compilation failed")
777                     exit()
778             else:
779                 print("At line", token[1], "an expression was expected")
780                 print("Compilation failed")
781                 exit()
782         else:
783             return [False]
```

Εικόνα 17

Κανόνας idTail:

- **Ελέγχος Αναγνωριστικού**: Αν το τρέχον token είναι αριστερή παρένθεση (leftpartoken), τότε ελέγχει αν ακολουθεί μια λίστα παραμέτρων (parList) και μετά αναμένει δεξιά παρένθεση (rightpartoken).

- **Επιστροφή Αποτελέσματος**: Επιστρέφει True αν η ουρά είναι συντακτικά σωστή, δηλαδή αν υπάρχει αριστερή και δεξιά παρένθεση μετά από μια λίστα παραμέτρων. Σε διαφορετική περίπτωση, επιστρέφει False.

```
785     def idTail(token):
786         if token[0] == "leftpartoken":
787             token = lex()
788             if parList(token):
789                 token = lex()
790                 if token[0] == "rightpartoken":
791                     return True
792                 else:
793                     print("At line", token[1], "a ')' was expected")
794                     print("Compilation failed")
795                     exit()
796             else:
797                 return False
798         else:
799             return False
```

Εικόνα 18

Κανόνας parList:

- **Ανάλυση Έκφρασης**: Αρχικά ελέγχει εάν υπάρχει μια έκφραση (expression). Αν ναι, τότε αποθηκεύει το αποτέλεσμα της έκφρασης σε μια μεταβλητή και προσθέτει την παράμετρο στην στοίβα (stack) με το κατάλληλο είδος (`cv` για το καλεσμένο τμήμα). Σημειώνει επίσης ότι τουλάχιστον μία παράμετρος υπάρχει.

- **Επανάληψη**: Εάν υπάρχει παράμετρος, η συνάρτηση επαναλαμβάνει τη διαδικασία για κάθε επόμενη παράμετρο που ακολουθεί μετά από το διαχωριστικό `,`.

- **Επιστροφή Αποτελέσματος**: Επιστρέφει True αν υπάρχουν παραμέτροι στη λίστα ή False αν δεν υπάρχει καμία παράμετρος.

```
801     def parList(token):
802         global seekIndex
803         global isAtLeastOnePar
804         global pos
805         global scope
806         expr1 = isExpression(token)
807         if expr1[0]:
808             E1place = expr1[1]
809             a = E1place
810             pos=pos+4
811             newPos=pos
812             newScope=scope-1
813             recordStructure.addNewEntity(scopeIndex=newScope,entityName=a,entity_type='Variable', offset=newPos)
814             quadList.append(genQuad("par",a,"CV","_"))
815             isAtLeastOnePar = True
816             seekIndex = f.tell()
817             token=lex()
818             while token[0] == "commatoken":
819                 seekIndex = f.tell()
820                 token = lex()
821                 expr2 = isExpression(token)
822                 if expr2[0]:
823                     E2place = expr2[1]
824                     b = E2place
825                     pos=pos+4
826                     newPos=pos
827                     newScope=scope-1
828                     recordStructure.addNewEntity(scopeIndex=newScope,entityName=b,entity_type='Variable', offset=newPos)
829                     quadList.append(genQuad("par",b,"CV","_"))
830                     seekIndex = f.tell()
831                     token=lex()
832                     continue
833                 else:
834                     print("At line", token[1], "an expression was expected after the ','")
835                     print("Compilation failed")
836                     exit()
837             f.seek(seekIndex)
838             return isAtLeastOnePar
839         else:
840             f.seek(seekIndex)
841             return True
```

Εικόνα 19

Κανόνας statement:

1. Αρχικά ελέγχει εάν η δήλωση είναι μια απλή δήλωση (simple statement) με τη χρήση της συνάρτησης `isSimpleStatement(token)`.
2. Αν η δήλωση δεν είναι απλή, ελέγχει εάν είναι δομημένη δήλωση (structured statement) με τη χρήση της συνάρτησης `isStructuredStatement(token)`.

Αν ο έλεγχος σε κάποια από τις δύο περιπτώσεις επιστρέψει `True`, τότε η συνάρτηση επιστρέφει επίσης `True`, δηλώνοντας ότι η είσοδος είναι μια δήλωση. Σε διαφορετική περίπτωση, επιστρέφει `False`.

```
843     def isStatement(token):
844         if isSimpleStatement(token):
845             return True
846         elif isStructuredStatement(token):
847             return True
848         else:
849             return False
850
```

Εικόνα 20

Κανόνας simple_statement:

1. Ελέγχει αρχικά εάν η δήλωση είναι μια ανάθεση με τη χρήση της συνάρτησης `isAssignmentStat(token)`.
2. Αν η δήλωση δεν είναι ανάθεση, ελέγχει εάν είναι μια εντολή εκτύπωσης με τη χρήση της συνάρτησης `isPrintStat(token)`.
3. Αν η δήλωση δεν είναι ούτε ανάθεση ούτε εντολή εκτύπωσης, ελέγχει εάν είναι μια εντολή επιστροφής με τη χρήση της συνάρτησης `isReturnStat(token)`.

Αν ο έλεγχος σε κάποια από τις παραπάνω περιπτώσεις επιστρέψει `True`, τότε η συνάρτηση επιστρέφει επίσης `True`, υποδεικνύοντας ότι η είσοδος είναι ένα απλό statement. Σε διαφορετική περίπτωση, επιστρέφει `False`.

```
851     def isSimpleStatement(token):
852         if isAssignmentStat(token):
853             return True
854         elif isPrintStat(token):
855             return True
856         elif isReturnStat(token):
857             return True
858         else:
859             return False
```

Εικόνα 21

Κανόνας assignment_stat:

• Έλεγχος για αναγνωριστικό (identifier): Αρχικά ελέγχει εάν η δήλωση ξεκινάει με ένα αναγνωριστικό, το οποίο αναγνωρίζεται από το token "anagnoristikotoken". Αν όχι, η δήλωση δεν είναι μια ανάθεση και η συνάρτηση επιστρέφει `False`.

• Έλεγχος για τον τελεστή ανάθεσης: Αν το πρώτο token είναι αναγνωριστικό, ελέγχει αν ακολουθεί ο τελεστής ανάθεσης "=" μετά το token "ana8eshtoken". Αν όχι, η δήλωση δεν είναι μια ανάθεση και η συνάρτηση επιστρέφει `False`.

• Έλεγχος για έκφραση: Αν οι προηγούμενοι έλεγχοι περάσουν, η συνάρτηση ελέγχει αν ακολουθεί μια έκφραση (expression) μετά τον τελεστή ανάθεσης. Αυτό γίνεται καλώντας τη συνάρτηση `isExpression(token)`. Αν υπάρχει έγκυρη έκφραση, δημιουργείται μια τριάδα (quadruple) για την ανάθεση με τη χρήση της συνάρτησης `genQuad(":=", Eplace, "_", idPlace)` και η συνάρτηση επιστρέφει `True`. Αν δεν υπάρχει έγκυρη έκφραση, εμφανίζεται μήνυμα σφάλματος και η συνάρτηση επιστρέφει `False`.

```
def isAssignmentStat(token):
    if token[0] == "anagnoristikotoken":
        idPlace = token[2]
        token = lex()
        if token[0] == "ana8eshtoken":
            token = lex()
            expr = isExpression(token)
            if expr[0]:
                Eplace = expr[1]
                quadList.append(genQuad(":=", Eplace, "_", idPlace))
                return True
            elif token[0] == "commandtoken":
                if token[2] == "int":
                    token = lex()
                    if token[0] == "leftpartoken":
                        token = lex()
                        if token[0] == "commandtoken":
                            if token[2] == "input":
                                token = lex()
                                if token[0] == "leftpartoken":
                                    token = lex()
                                    if token[0] == "rightpartoken":
                                        token = lex()
                                        if token[0] == "rightpartoken":
                                            quadList.append(genQuad("inp",idPlace,"_","_"))
                                            return True
                                        else:
                                            print("At line", token[1], "a ')' was expected")
                                            print("Compilation failed")
                                            exit()
                                    else:
                                        print("At line", token[1], "a ')' was expected")
                                        print("Compilation failed")
                                        exit()
                                else:
                                    print("At line", token[1], "a '(' was expected")
                                    print("Compilation failed")
                                    exit()
                            else:
                                return False
                        else:
                            return False
                    else:
                        print("At line", token[1], "a '(' was expected")
                        print("Compilation failed")
                        exit()
                else:
                    return False
```

Εικόνα 22

- Πρόσθετος Έλεγχος για 'int(input())': Αν η έκφραση είναι εσφαλμένη και το επόμενο token είναι η εντολή "int", η συνάρτηση προχωράει σε ένα επιπρόσθετο έλεγχο για να επιβεβαιώσει ότι η δήλωση είναι της μορφής "int(input())". Αν αυτό ισχύει, δημιουργείται μια τριάδα για την είσοδο με τη χρήση της συνάρτησης `genQuad("inp", idPlace, "_", "_")` και η συνάρτηση επιστρέφει `True`. Αν δεν είναι της αναμενόμενης μορφή

```
                else:
                    print("At line", token[1], "either an expression or 'int(input()) was expected after the '='")
                    print("Compilation failed")
                    exit()
            else:
                return False
        else:
            return False
```

Εικόνα 23

Κανόνας return_stat:

• Έλεγχος για την εντολή επιστροφής: Αρχικά ελέγχει εάν το πρώτο token είναι η εντολή "return", το οποίο αναγνωρίζεται από το token "commandtoken". Αν δεν είναι, η συνάρτηση επιστρέφει `False`.

• Έλεγχος για έκφραση: Αν το πρώτο token είναι η εντολή "return", τότε ελέγχει εάν ακολουθεί μια έκφραση (expression). Αυτό γίνεται καλώντας τη συνάρτηση `isExpression(token)`. Αν υπάρχει έγκυρη έκφραση, δημιουργείται μια τριάδα (quadruple) για την επιστροφή με τη χρήση της συνάρτησης `genQuad("ret", Eplace, "_", "_")` και η συνάρτηση επιστρέφει `True`. Αν δεν υπάρχει έγκυρη έκφραση, εμφανίζεται μήνυμα σφάλματος και η συνάρτηση επιστρέφει `False`.

```
def isReturnStat(token):
    if token[0] == "commandtoken":
        if token[2] == "return":
            token = lex()
            expr = isExpression(token)
            if expr[0]:
                Eplace = expr[1]
                quadList.append(genQuad("ret", Eplace, "_", "_"))
                return True
            else:
                print("At line", token[1], "an expression was expected after the 'return' command")
                print("Compilation failed")
                exit()
        else:
            return False
    else:
        return False
```

Εικόνα 24

Κανόνας structure_statement:

• Έλεγχος για δήλωση if: Καλεί τη συνάρτηση `ifState(token)` για να ελέγξει εάν το token αντιπροσωπεύει μια δήλωση if. Αν ναι, επιστρέφει `True`.

• Έλεγχος για δήλωση while: Καλεί τη συνάρτηση `whileState(token)` για να ελέγξει εάν το token αντιπροσωπεύει μια δήλωση while. Αν ναι, επιστρέφει `True`.

• Επιστροφή False: Αν το token δεν αντιστοιχεί ούτε σε δήλωση if ούτε σε δήλωση while, επιστρέφει `False`.

```
def isStructuredStatement(token):

    if ifState(token):
        return True
    elif whileState(token):
        return True
    else:
        return False
```

Εικόνα 25

Κανόνας while_stat:

• Έλεγχος εάν το token αντιπροσωπεύει τη δήλωση `while`: Ελέγχει πρώτα εάν το token είναι έγκυρη εντολή (command token) και αν αντιστοιχεί στη λέξη-κλειδί "while". Αν όχι, επιστρέφει `False`.

• Ανάλυση του σώματος της while λούπας: Αφού ελέγξει ότι η δήλωση ξεκινά με τη λέξη-κλειδί "while", συνεχίζει τον έλεγχο με την εξής σειρά:

- Ελέγχει τη συνθήκη της while λούπας χρησιμοποιώντας τη συνάρτηση `isCondition(token)`.
- Εάν η συνθήκη είναι έγκυρη, τότε ελέγχει αν ακολουθείται από το σωστό σύμβολο '{' με τη συνάρτηση `lex()`.
- Ακολουθεί έλεγχος για τα statements που βρίσκονται μέσα στο σώμα της while λούπας χρησιμοποιώντας τη συνάρτηση `statements(token)`.
- Ελέγχει εάν το σώμα της while λούπας κλείνει με το κατάλληλο σύμβολο '}'.
- Εάν όλα τα παραπάνω είναι έγκυρα, επιστρέφει `True`.

• Επιστροφή False: Εάν το token δεν αντιπροσωπεύει μια δήλωση while, τότε επιστρέφει `False`.

```
945     def whileState(token):
946         if token[0] == "commandtoken":
947             if token[2] == "while":
948                 condQuad = nextQuad()
949                 token = lex()
950                 cond = isCondition(token)
951                 if cond[0]:
952                     conditionTrue = cond[1]
953                     conditionFalse = cond[2]
954                     backpatch(conditionTrue, nextQuad())
955                     token = lex()
956                     if token[0] == "anwkatwtoken":
957                         token = lex()
958                         if token[0] == "anoigmatoken":
959                             token = lex()
960                             if statements(token):
961                                 token = lex()
962                                 if token[0] == "kleisimotoken":
963 
964                                     quadList.append(genQuad("jump","_","_",condQuad))
965                                     backpatch(conditionFalse,nextQuad())
966                                     return True
967                                 else:
968                                     print("At line", token[1] ,"a '#}' was expected to close a '#{ found earlier for the while loop")
969                                     print("Compilation failed")
970                                     exit()
971                             else:
972                                 print("At line", token[1], "a statement was expected after the ':'")
973                                 print("Compilation failed")
974                                 exit()
975                         else:
976                             print("At line", token[1], "a '#{' was expected after the 'while'")
977                             print("Compilation failed")
978                             exit()
979                     else:
980                         print("At line", token[1], "a ':' was expected")
981                         print("Compilation failed")
982                         exit()
983                 else:
984                     print("At line", token[1], "a condition was expected after the 'while' command")
985                     print("Compilation failed")
986                     exit()
987             else:
988                 return False
989         else:
990             return False
991 
```

Εικόνα 26

Κανόνας startRule:

- **Λήψη επόμενου token**: Αρχικά, καλεί τη συνάρτηση `lex()` για να λάβει το πρώτο token του προγράμματος.

- **Δημιουργία νέου επιπέδου εμβέλειας**: Προσθέτει ένα νέο επίπεδο εμβέλειας στη δομή εγγραφής για τις μεταβλητές και τις συναρτήσεις.

- **Έλεγχος δηλώσεων**: Ελέγχει εάν υπάρχουν δηλώσεις (declarations) στο πρόγραμμα χρησιμοποιώντας τη συνάρτηση `declarations(token)`. Εάν η διαδικασία δηλώσεων είναι επιτυχής, συνεχίζει, διαφορετικά τερματίζει τη μεταγλώττιση.

- **Έλεγχος και κλήση συναρτήσεων**: Ελέγχει και καλεί τη συνάρτηση `isDefFunctions(token)` για κάθε επόμενο token, έως ότου δεν υπάρχουν άλλες δηλώσεις συναρτήσεων στο πρόγραμμα.

- **Έλεγχος και κλήση του κύριου τμήματος (main)**: Ελέγχει εάν υπάρχει το κύριο τμήμα (main) του προγράμματος με τη συνάρτηση `callMainPart(token)`. Εάν όλα είναι έγκυρα, προσθέτει τις αντίστοιχες τελικές τετράδες (`halt`, `end_block`), εμφανίζει την τελική τετράδα και επιστρέφει `True`. Εάν δεν βρεθεί το κύριο τμήμα, το πρόγραμμα τερματίζει με σφάλμα.

```
992     def startRule():
993         token=lex()
994         recordStructure.addNewScope()
995         if declarations(token):
996             token=lex()
997             while isDefFunctions(token):
998                 token=lex()
999             if callMainPart(token):
1000                quadList.append(genQuad("halt","_","_","_"))
1001                quadList.append(genQuad("end_block","main","_","_"))
1002                readQuadList(quadList[-1])
1003                return True
1004            else:
1005                exit()
1006        else:
1007            print("At line", token[1]," declarations was expected")
1008            print("Compilation failed")
1009            exit()
```

Εικόνα 27

Κανόνας print_stat:

1. **Έλεγχος τύπου εντολής**: Αρχικά, ελέγχει εάν το τρέχον token είναι εντολή. Αν δεν είναι, επιστρέφει `False`.

2. **Έλεγχος εντολής εκτύπωσης**: Αν το τρέχον token είναι εντολή εκτύπωσης (`print`), συνεχίζει τον έλεγχο, αλλιώς επιστρέφει `False`.
3. **Έλεγχος αριστερού αγκύλης**: Περιμένει το επόμενο token να είναι η αριστερή αγκύλη (`(`). Αν δεν είναι, εμφανίζει ένα μήνυμα σφάλματος και τερματίζει τη μεταγλώττιση.
4. **Έλεγχος έκφρασης εκτύπωσης**: Αναμένει μια έκφραση μετά το αριστερό αγκύλη. Αν υπάρχει έκφραση, συνεχίζει τον έλεγχο, αλλιώς εμφανίζει ένα μήνυμα σφάλματος και τερματίζει τη μεταγλώττιση.
5. **Έλεγχος δεξιού αγκύλης**: Αναμένει τη δεξιά αγκύλη (`)`) μετά την έκφραση. Αν αυτό δε συμβαίνει, εμφανίζει ένα μήνυμα σφάλματος και τερματίζει τη μεταγλώττιση.

Εάν όλα τα βήματα περάσουν επιτυχώς, προσθέτει μια τετράδα κατάλληλης εντολής εκτύπωσης (`out`) στην τελική λίστα τετράδων και επιστρέφει `True`.

```
1011    def isPrintStat(token):
1012        global seekIndex
1013        if token[0] == "commandtoken":
1014            if token[2] == "print":
1015                token = lex()
1016                if token[0] == "leftpartoken":
1017                    token = lex()
1018                    expr = isExpression(token)
1019                    if expr[0]:
1020                        Eplace = expr[1]
1021                        token = lex()
1022                        if token[0] == "rightpartoken":
1023                            quadList.append(genQuad("out",Eplace,"_","_"))
1024                            seekIndex = f.tell()
1025                            return True
1026                        else:
1027                            print("At line", token[1]," a ')' was expected")
1028                            print("Compilation failed")
1029                            exit()
1030                    else:
1031                        print("At line", token[1], "an expression was expected after the 'print' command")
1032                        print("Compilation failed")
1033                        exit()
1034                else:
1035                    print("At line", token[1]," a '(' was expected after the 'print' command")
1036                    print("Compilation failed")
1037                    exit()
1038            else:
1039                return False
1040        else:
1041            return False
```

Εικόνα 28

Κανόνας id_List:

• **Αρχικοποίηση μεταβλητών**: Αρχικά, ορίζονται δύο μεταβλητές, η `seekIndex` και η `isAtLeastOneID`.

• **Έλεγχος αναγνωριστικού**: Αν το πρώτο token είναι αναγνωριστικό, θεωρείται ότι υπάρχει τουλάχιστον ένα αναγνωριστικό στη λίστα και καταγράφεται η θέση του στη μεταβλητή `seekIndex`. Το αναγνωριστικό προστίθεται στη λίστα `names`.

• **Επανάληψη για περισσότερα αναγνωριστικά**: Αν ακολουθεί κόμμα, ελέγχει αν ακολουθεί ένα ακόμα αναγνωριστικό. Αν ναι, το προσθέτει στη λίστα `names` και

καταγράφει τη θέση του με το `seekIndex`. Επαναλαμβάνει αυτή τη διαδικασία μέχρι να τελειώσει η λίστα αναγνωριστικών.

- **Επιστροφή αποτελεσμάτων**: Επιστρέφει μια λίστα που περιέχει ένα λογικό boolean (`isAtLeastOneID`) που δηλώνει εάν υπάρχει τουλάχιστον ένα αναγνωριστικό και μια λίστα (`names`) με τα αναγνωριστικά που βρέθηκαν.

```
def isIdList(token):
    global seekIndex
    global isAtLeastOneID
    global pos
    global scope
    names =[]
    if token[0]=="anagnoristikotoken":
        isAtLeastOneID = True
        seekIndex = f.tell()
        names.append(token[2])
        token=lex()
        newScope=scope-1
        while token[0]=="commatoken":
            token=lex()
            if token[0]=="anagnoristikotoken":
                names.append(token[2])
                seekIndex = f.tell()
                token=lex()
            else:
                print("At line", token[1]," an id was expected")
                print("Compilation failed")
                exit()
        f.seek(seekIndex)
        for x in names:
            pos=pos+4
            newPos=pos
            recordStructure.addNewEntity(scopeIndex=newScope,entityName=x,entity_type='Variable', offset=newPos)
        return [isAtLeastOneID,names]
    else:
        f.seek(seekIndex)
        return [False,names]
```

Εικόνα 29

Κανόνας statements:

- **Έλεγχος δήλωσης**: Αρχικά, ελέγχει εάν το πρώτο token αντιπροσωπεύει μια δήλωση. Αν όχι, επιστρέφει `False`.

- **Επανάληψη για περισσότερες δηλώσεις**: Αν το πρώτο token αντιπροσωπεύει μια δήλωση, τότε εκτελεί μια επανάληψη για να ελέγξει αν υπάρχουν κι άλλες δηλώσεις στη σειρά. Αν όχι, επιστρέφει `True`.

- **Επιστροφή αποτελέσματος**: Επιστρέφει `True` αν όλες οι δηλώσεις είναι έγκυρες, αλλιώς `False`.

```
1068    def statements(token):
1069        global seekIndex
1070        if isStatement(token) == False:
1071            f.seek(seekIndex)
1072            return False
1073        f.seek(seekIndex)
1074        token = lex()
1075        while isStatement(token):
1076            seekIndex = f.tell()
1077            f.seek(seekIndex)
1078            token=lex()
1079
1080        f.seek(seekIndex)
1081        return True
```

Εικόνα 30

Κανόνας declarations:

- **Έλεγχος δήλωσης**: Αρχικά, ελέγχει εάν το πρώτο token αντιπροσωπεύει μια δήλωση. Αν υπάρχει τουλάχιστον μία δήλωση, η μεταβλητή `isAtLeastOneDeclaration` θέτεται σε `True`.

- **Επανάληψη για περισσότερες δηλώσεις**: Αν υπάρχει τουλάχιστον μία δήλωση και η μεταβλητή `isAtLeastOneDeclaration` είναι `True`, τότε εκτελείται μια επανάληψη για να ελεγχθούν και οι υπόλοιπες δηλώσεις.

- **Επιστροφή αποτελέσματος**: Επιστρέφει την τιμή της μεταβλητής `isAtLeastOneDeclaration`, η οποία δείχνει εάν υπάρχει τουλάχιστον μία έγκυρη δήλωση ή όχι.

```
1083    def declarations(token):
1084        global seekIndex
1085        global isAtLeastOneDeclaration
1086        seekIndex = f.tell()
1087        isAtLeastOneDeclaration = isDeclaration(token)
1088        token=lex()
1089        while isDeclaration(token) and isAtLeastOneDeclaration:
1090            seekIndex = f.tell()
1091            token = lex()
1092        f.seek(seekIndex)
1093        return isAtLeastOneDeclaration
```

Εικόνα 31

Κανόνας globals:

- **Έλεγχος δήλωσης**: Αρχικά, ελέγχει εάν το πρώτο token αντιπροσωπεύει μια δήλωση για καθολική μεταβλητή. Αν υπάρχει τουλάχιστον μία δήλωση, η μεταβλητή `isAtLeastOneGlobal` θέτεται σε `True`.

- **Επανάληψη για περισσότερες δηλώσεις**: Αν υπάρχει τουλάχιστον μία δήλωση και η μεταβλητή `isAtLeastOneGlobal` είναι `True`, τότε εκτελείται μια επανάληψη για να ελεγχθούν και οι υπόλοιπες δηλώσεις για καθολικές μεταβλητές.

- **Επιστροφή αποτελέσματος**: Επιστρέφει την τιμή της μεταβλητής `isAtLeastOneGlobal`, η οποία δείχνει εάν υπάρχει τουλάχιστον μία έγκυρη δήλωση για καθολική μεταβλητή ή όχι.

```
1095    def globalVar(token):
1096        global seekIndex
1097        global isAtLeastOneGlobal
1098        seekIndex = f.tell()
1099        isAtLeastOneGlobal = isGlobal(token)
1100        token=lex()
1101        while isGlobal(token):
1102            seekIndex = f.tell()
1103            token=lex()
1104        f.seek(seekIndex)
1105        return isAtLeastOneGlobal
```

Εικόνα 32

Κανόνας declaration:

- **Έλεγχος τύπου δήλωσης**: Αρχικά, ελέγχει εάν το πρώτο token είναι ένα `intdeftoken`, που υποδηλώνει δήλωση μεταβλητής τύπου `int`.

- **Ανάλυση της idList**: Αν η δήλωση είναι τύπου `int`, συνεχίζει τον έλεγχο με τη λειτουργία `isIdList` για να ελέγξει αν ακολουθείται από μια έγκυρη λίστα ταυτοποιητών.

• **Επιστροφή αποτελέσματος**: Επιστρέφει `True` αν η δήλωση είναι έγκυρη, δηλαδή αν ξεκινά με `intdeftoken` και ακολουθείται από μια έγκυρη λίστα ταυτοποιητών, αλλιώς επιστρέφει `False`.

```
1107    def isDeclaration(token):
1108        if token[0]=="intdeftoken":
1109            token=lex()
1110            idList =isIdList(token)
1111            if idList[0]:
1112                return True
1113            else:
1114                print("At line", token[1], "an id was expected ")
1115                print("Compilation failed")
1116                exit()
1117        else:
1118            return False
1119
```

Εικόνα 33

Κανόνας global:

• **Έλεγχος είδους δήλωσης**: Αρχικά, ελέγχει εάν το πρώτο token είναι ένα `commandtoken` και εάν η εντολή είναι `global`.

• **Ανάλυση της idList**: Αν η εντολή είναι τύπου `global`, συνεχίζει τον έλεγχο με τη λειτουργία `isIdList` για να ελέγξει αν ακολουθείται από μια έγκυρη λίστα ταυτοποιητών.

• **Επιστροφή αποτελέσματος**: Επιστρέφει `True` αν η εντολή είναι έγκυρη, δηλαδή αν ξεκινά με `global` και ακολουθείται από μια έγκυρη λίστα ταυτοποιητών, αλλιώς επιστρέφει `False`.

```
1120    def isGlobal(token):
1121        if token[0]=="commandtoken":
1122            if token[2]=="global":
1123                token=lex()
1124                idList =isIdList(token)
1125                if idList[0]:
1126                    return True
1127                else:
1128                    print("At line", token[1], "an id was expected ")
1129                    print("Compilation failed")
1130                    exit()
1131            else:
1132                return False
1133        else:
1134            return False
```

Εικόνα 34

Κανόνας def_function:

- **Έλεγχος του πρώτου token:**
    - Ξεκινάει ελέγχοντας εάν το πρώτο token είναι ένα `commandtoken` και εάν η εντολή είναι `def`. Αυτό υποδηλώνει ότι βρίσκεται μια δήλωση συνάρτησης.

- **Ανάλυση του ονόματος της συνάρτησης** :
    - Αν το τρέχον token είναι `def`, τότε περιμένουμε να ακολουθήσει το όνομα της συνάρτησης (ένα `anagnoristikotoken`). Αυτό το όνομα αποθηκεύεται στην μεταβλητή `q`.
    - Στη συνέχεια, περιμένουμε ένα `leftpartoken`, δείχνοντας την αρχή της λίστας των ορισμάτων της συνάρτησης.
    - Ακολουθεί η ανάλυση της λίστας των ορισμάτων της συνάρτησης, η οποία γίνεται με την κλήση της συνάρτησης `isIdList`.
    - Στη συνέχεια, αποθηκεύουμε το μήκος του πλαισίου της συνάρτησης (το οποίο υπολογίζεται με βάση τον αριθμό των ορισμάτων) στη μεταβλητή `framelen`.
    - Επιπλέον, δημιουργούμε μια νέα εγγραφή στη δομή δεδομένων του προγράμματος για την αποθήκευση της συνάρτησης, συμπεριλαμβανομένων των ορισμάτων της.

```
1136    def isDefFunctions(token):
1137        global scope
1138        global pos
1139        if token[0] == "commandtoken":
1140            if token[2] == "def":
1141                token=lex()
1142                if token[0] == "anagnoristikotoken":
1143                    lastBlock.append(token[2])
1144                    q = token[2]
1145                    pos=pos+4
1146                    newPos=pos
1147                    newScope=scope-1
1148                    token=lex()
1149                    if token[0] == "leftpartoken":
1150                        token = lex()
1151                        idList =isIdList(token)
1152                        if idList[0]:
1153                            framelen=newPos
1154                            recordStructure.addNewEntity(scopeIndex=newScope,entityName=q,entity_type='Function', framelength=newPos,arguments=idList[1])
1155                            token=lex()
1156                            if token[0] == "rightpartoken":
1157                                token = lex()
1158                                if token[0] == "anwkatwtoken":
1159                                    recordStructure.addNewScope()
1160                                    token = lex()
1161                                    if token[0] == "anoigmatoken":
1162                                        token = lex()
1163                                        if declarations(token):
1164                                            token = lex()
1165                                        hasFunction = False
1166                                        while True:
1167                                            if isDefFunctions(token):
1168                                                hasFunction = True
1169                                                token = lex()
1170                                                continue
1171                                            else:
1172                                                if hasFunction == False:
1173                                                    quadList.append(emptyList())
1174                                                    indexList.append(len(quadList)-1)
1175                                                break
```

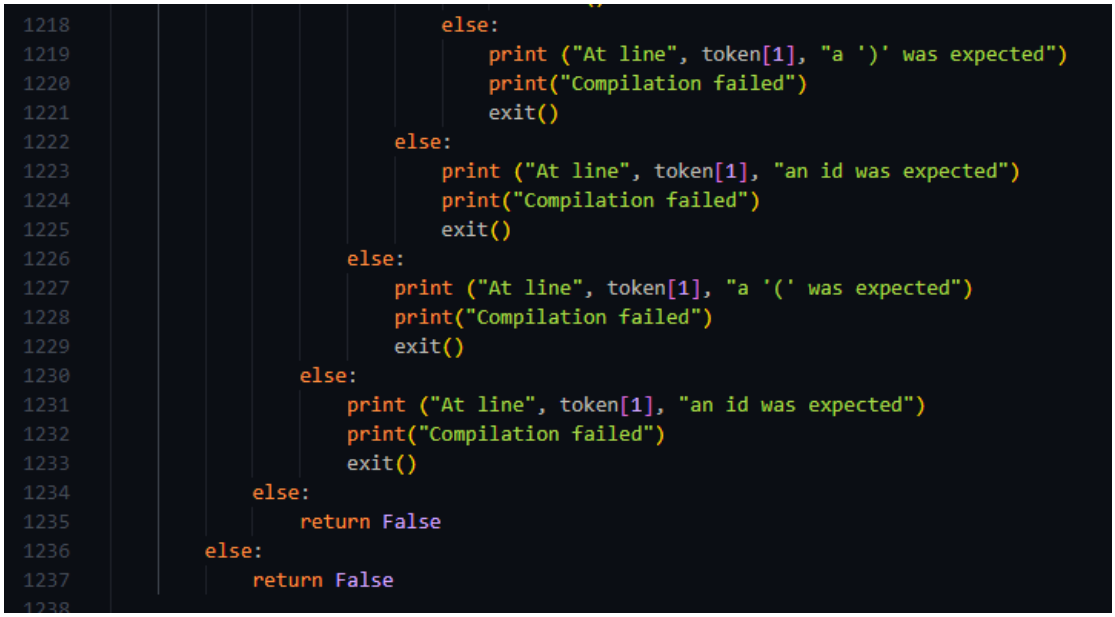
Εικόνα 35

```
                        if statements(token):
                            token = lex()

                            if token[0] == "kleisimotoken":
                                quadList.append(genQuad("end_block",lastBlock[-1],"_","_"))

                                pos=newPos
                                if len(lastBlock) == 1 and len(indexList) == 0:
                                    lastBlock.pop()
                                    readQuadList(quadList[-1])
                                    return True
                                if len(indexList)>0:
                                    quadList[indexList[-1]] = ["begin_block",lastBlock.pop(),"_","_"]

                                    indexList.pop()
                                    readQuadList(quadList[-1])
                                    if len(indexList) == 0 and len(lastBlock)>0:

                                        quadList.append(genQuad("begin_block",lastBlock[-1],"_","_"))
                                else:

                                    quadList.append(genQuad("begin_block",lastBlock[-1],"_","_"))

                                    return True
                                else:
                                    print ("At line", token[1], "a '#}' was expected to close a '#{' found earlier")
                                    print("Compilation failed")
                                    exit()
                            else:
                                print ("At line", token[1], "a statement was expected")
                                print("Compilation failed")
                                exit()
                        else:
                            print ("At line", token[1], "a '#{' was expected")
                            print("Compilation failed")
                            exit()
                    else:
                        print ("At line", token[1], "a ':' was expected")
                        print("Compilation failed")
                        exit()
                else:
                    print ("At line", token[1], "a ')' was expected")
                    print("Compilation failed")
                    exit()
```

Εικόνα 36

- **Ανάλυση του σώματος της συνάρτησης** :
    - Ακολουθεί η ανάλυση του σώματος της συνάρτησης.
    - Αναλύονται οι δηλώσεις με τη χρήση της συνάρτησης `declarations`.
    - Ελέγχεται η ύπαρξη άλλων ενσωματωμένων συναρτήσεων με τη χρήση της `isDefFunctions`.
    - Ακολουθεί η ανάλυση των μεταβλητών (global variables) με τη χρήση της `globalVar`.
    - Τέλος, ακολουθεί η ανάλυση των εντολών του σώματος της συνάρτησης με τη χρήση της `statements`.

- **Ολοκλήρωση της ανάλυσης** :
    - Ολοκληρώνει την ανάλυση του σώματος της συνάρτησης.
    - Προσθέτει τα απαραίτητα τελικά τετράδια (`genQuad`) για την έξοδο.



Εικόνα 37

- **Προσθήκη των τελικών τετράδων** :

  - Προστίθενται τα τελικά τετράδια για τον τερματισμό της συνάρτησης.
  - Αν υπάρχει μόνο ένα επίπεδο συναρτήσεων και δεν υπάρχουν ενσωματωμένες συναρτήσεις (`isDefFunctions`), τότε παίρνεται η εντολή `halt` για τον τερματισμό του προγράμματος.
  - Εισάγεται το τελικό τετράδιο `end_block` για να δηλώσει το τέλος του σώματος της συνάρτησης.

- **Επιστροφή αποτελέσματος** :

  - Επιστρέφει `True` αφού ολοκληρώθηκε επιτυχώς η ανάλυση της συνάρτησης.
  - Εάν κάποια από τις αναμενόμενες εντολές λείπει, το πρόγραμμα εκτυπώνει ένα μήνυμα σφάλματος και τερματίζεται.

Κανόνας call_main_part:

- **Έλεγχος ορισμού της συνάρτησης `main`**: Αρχικά, ελέγχει εάν το πρώτο token είναι `defitoken` και αν η συνάρτηση έχει όνομα `main`.

- **Δημιουργία εγγραφής συνάρτησης**: Αν η συνάρτηση `main` είναι έγκυρη, προσθέτει μια εγγραφή για τη συνάρτηση `main` στη δομή δεδομένων `recordStructure`. Η εγγραφή αυτή περιλαμβάνει το όνομα, το μήκος του πλαισίου (framelength), το οποίο στην περίπτωσή σας είναι το μέγεθος της κύριας δομής δεδομένων `scope` (`framelen`), καθώς και τη λίστα των ορισμάτων (κενή για την `main`).

- **Δημιουργία του πρώτης τετράδας**: Προσθέτει μια ειδική τετράδα που δηλώνει την έναρξη του κύριου τμήματος κώδικα.

• **Έλεγχος δηλώσεων και εντολών**: Ελέγχει τις δηλώσεις και τις εντολές που ακολουθούν τον ορισμό της συνάρτησης `main`.

• **Επιστροφή αποτελέσματος**: Επιστρέφει `True` εάν ο ορισμός της συνάρτησης `main` είναι έγκυρος και ακολουθούν έγκυρες δηλώσεις και εντολές, αλλιώς επιστρέφει `False`.

```
1240    def callMainPart(token):
1241        global scope
1242        global pos
1243        global framelen
1244        recordStructure.addNewScope()
1245        if token[0] == "defitoken":
1246            token = lex()
1247            if token[0] == "commandtoken":
1248                if token[2] == "main":
1249                    q = token[2]
1250                    pos=pos+4
1251                    newPos=pos
1252                    newScope=scope-1
1253                    framelen=newScope
1254                    recordStructure.addNewEntity(scopeIndex=newScope,entityName=q,entity_type='Function', framelength=newPos,arguments=[])
1255                    quadList.append(genQuad("begin_block","main","_","_"))
1256                    token=lex()
1257                    if declarations(token):
1258                        token=lex()
1259                    if statements(token):
1260                            return True
1261                    else:
1262                        print ("At line", token[1], "a statement was expected")
1263                        print("Compilation failed")
1264                        exit()
1265                else:
1266                    exit()
1267            else:
1268                exit()
1269        else:
1270            return False
```

Εικόνα 38

5)Ενδιάμεσος κώδικας

Για την υλοποίηση του ενδιάμεσου κώδικα χρησιμοποιήθηκαν οι βοηθητικές συναρτήσεις (η υλοποίηση τους φαίνεται στις Εικόνα 6) που περιγράφονται στις διαφάνειες και το handbook του μαθήματος και τοποθετήθηκαν στα κατάλληλα σημεία με τις κατάλληλες παραμέτρους (σύμφωνα με το handbook), ώστε να παραχθεί ο ενδιάμεσος κώδικας. .Η περιγραφή των παραπάνω φαίνεται στο κεφάλαιο 4 αναλυτικά.

```
1297        def nextQuad():
1298            global quadNum
1299            return quadNum+1
1300
1301        def genQuad(op,x,y,z):
1302            global quadNum
1303            quadNum += 1
1304            quad = [op,x,y,z]
1305            return quad
1306
1307        def newTemp():
1308            global tempNum
1309            tempNum += 1
1310            s = "T_"
1311            return s+str(tempNum)
1312
1313        def emptyList():
1314            return genQuad("_","_","_","_")
1315
1316        def makeList(label):
1317            return [label]
1318
1319        def mergeList(list1,list2):
1320            return list1+list2
1321
1322        def backpatch(lst, label):
1323            for x in lst:
1324                quadList[x-1][-1] = label
1325
```

Εικόνα 39

6)Πίνακας συμβόλων

Για τον πίνακα συμβόλων δημιουργήθηκε η κλάση RecordStructure (Εικόνες 40,41&42) που υλοποιεί την δομή (σύμφωνα με τις διαφάνειες και το handbook) για τον πίνακα συμβόλων. Με την χρήση τον μεθόδων αυτής της δομής στα κατάλληλα σημεία δημιουργείται ο πίνακας. Πιο συγκεκριμένα χρησιμοποιείται η global μεταβλητή scope που καθορίζει το scope του πίνακα και pos που καθορίζει το offset . Το pos αυξάνεται κατά 4 όταν δημιουργείται κάποιο entity. Το scope αυξάνεται κατά 1 όταν δημιουργείται κάποια συνάρτηση και μειώνεται κατά 1 όταν φτάνει στο τέλος της .Η περιγραφή των παραπάνω φαίνεται στο κεφάλαιο 4 αναλυτικά.

```python
class RecordStructure:
    def __init__(self):
        self.scopes = []

    class RecordScope:
        def __init__(self, entityList, nestingLevel):
            self.entityList = entityList
            self.nestingLevel = nestingLevel

    class RecordEntity:
        def __init__(self, name):
            self.name = name

    class Variable(RecordEntity):
        def __init__(self, name,  offset):
            super().__init__(name)
            self.offset = offset

    class Function(RecordEntity):
        def __init__(self, name,  framelength,arguments):
            super().__init__(name)
            self.framelength = framelength
            self.arguments = arguments

    class TemporaryVariable(RecordEntity):
        def __init__(self, name, offset):
            super().__init__(name)
            self.offset = offset

    class RecordArgument:
        def __init__(self, parMode):
            self.parMode = parMode
```

Εικόνα 40

Εσωτερικές Κλάσεις

1. **`RecordScope`**:
   - Αναπαριστά ένα πεδίο (scope) με μια λίστα από οντότητες (`entityList`) και ένα επίπεδο φωλιάσματος (`nestingLevel`).
   - `__init__(self, entityList, nestingLevel)`: Κατασκευαστής που αρχικοποιεί τη λίστα οντοτήτων και το επίπεδο φωλιάσματος.
2. **`RecordEntity`**:
   - Η βάση κλάση για όλες τις οντότητες, περιέχει μόνο το όνομα της οντότητας.
   - `__init__(self, name)`: Κατασκευαστής που αρχικοποιεί το όνομα.
3. **`Variable`** (κληρονομεί από `RecordEntity`):

- Αναπαριστά μια μεταβλητή με ένα όνομα και μια μετατόπιση (`offset`).
- `__init__(self, name, offset)`: Κατασκευαστής που αρχικοποιεί το όνομα και τη μετατόπιση.

4. **`Function`** (κληρονομεί από `RecordEntity`):
   - Αναπαριστά μια συνάρτηση με ένα όνομα, μήκος πλαισίου (`framelength`) και λίστα επιχειρημάτων (`arguments`).
   - `__init__(self, name, framelength, arguments)`: Κατασκευαστής που αρχικοποιεί το όνομα, το μήκος πλαισίου και τα επιχειρήματα.
5. **`TemporaryVariable`** (κληρονομεί από `RecordEntity`):
   - Αναπαριστά μια προσωρινή μεταβλητή με ένα όνομα και μια μετατόπιση (`offset`).
   - `__init__(self, name, offset)`: Κατασκευαστής που αρχικοποιεί το όνομα και τη μετατόπιση.
6. **`RecordArgument`**:
   - Αναπαριστά ένα επιχείρημα μιας συνάρτησης με μια λειτουργία παραμέτρου (`parMode`).
   - `__init__(self, parMode)`: Κατασκευαστής που αρχικοποιεί τη λειτουργία παραμέτρου.

```
174     def addNewScope(self):
175         newScope = RecordStructure.RecordScope(entityList=[], nestingLevel=len(self.scopes) + 1)
176         self.scopes.append(newScope)
177         return newScope
178 
179     def scopeDeletion(self, scopeIndex):
180         if scopeIndex < len(self.scopes):
181             self.scopes.pop(scopeIndex)
182 
183 
184     def addNewEntity(self, scopeIndex, entityName, entity_type, **kwargs):
185         if scopeIndex < len(self.scopes):
186             if entity_type == 'Variable':
187                 newEntity = RecordStructure.Variable(name=entityName, **kwargs)
188             elif entity_type == 'Function':
189                 newEntity = RecordStructure.Function(name=entityName, **kwargs)
190             elif entity_type == 'TemporaryVariable':
191                 newEntity = RecordStructure.TemporaryVariable(name=entityName, **kwargs)
192             else:
193                 return None
194             self.scopes[scopeIndex].entityList.append(newEntity)
195             return newEntity
196 
197     def addNewArgument(self, scopeIndex, entityIndex, argumentName, parMode, type):
198         if scopeIndex < len(self.scopes) and entityIndex < len(self.scopes[scopeIndex].entityList):
199             newArgument = RecordStructure.RecordArgument(parMode=parMode, type=type)
200             if not hasattr(self.scopes[scopeIndex].entityList[entityIndex], "arguments"):
201                 self.scopes[scopeIndex].entityList[entityIndex].arguments = []
202             self.scopes[scopeIndex].entityList[entityIndex].arguments.append(newArgument)
203             return newArgument
204 
205     def searchEntity(self, entityName):
206         for scope in self.scopes:
207             for entity in scope.entityList:
208                 if entity.name == entityName:
209                     return entity, None
210         return None, "Entity not found."
211 
```

Εικόνα 41

Μέθοδοι της `RecordStructure`

1. **`__init__(self)`**:
   - Αρχικοποιεί μια κενή λίστα από πεδία (`scopes`).
2. **`addNewScope(self)`**:
   - Προσθέτει ένα νέο πεδίο στην λίστα των πεδίων.

- Επιστρέφει το νέο πεδίο.

3. **`scopeDeletion(self, scopeIndex)`**:
   - Διαγράφει ένα πεδίο από την λίστα των πεδίων σύμφωνα με τον δείκτη του.
4. **`addNewEntity(self, scopeIndex, entityName, entity_type, **kwargs)`**:
   - Προσθέτει μια νέα οντότητα σε ένα συγκεκριμένο πεδίο.
   - `entity_type` μπορεί να είναι 'Variable', 'Function' ή 'TemporaryVariable'.
   - Επιστρέφει τη νέα οντότητα.
5. **`addNewArgument(self, scopeIndex, entityIndex, argumentName, parMode, type)`**:
   - Προσθέτει ένα νέο επιχείρημα σε μια συνάρτηση μέσα σε ένα συγκεκριμένο πεδίο και οντότητα.
   - Επιστρέφει το νέο επιχείρημα.
6. **`searchEntity(self, entityName)`**:
   - Αναζητά μια οντότητα σύμφωνα με το όνομα της σε όλα τα πεδία.
   - Επιστρέφει την οντότητα και το επίπεδο φωλιάσματος (ή `None` αν δεν βρεθεί).
7. **`printScopesToFile(self, filename="cpy.sym")`**:
   - Εκτυπώνει τα πεδία και τις οντότητες σε ένα αρχείο.
   - Δημιουργεί ένα αρχείο με αναφορά σε κάθε πεδίο και τις οντότητες που περιέχει.

```
212     def printScopesToFile(self, filename="cpy.sym"):
213         with open(filename, "w") as file:
214             file.write("--------------------------------SYMBOL TABLE--------------------------------\n")
215             for i, scope in enumerate(self.scopes, start=1):
216                 file.write(f"Scope {i-1}:\n")
217                 #file.write(f"Nesting Level: {scope.nestingLevel}\n")
218                 file.write("Entities:\n")
219                 for j, entity in enumerate(scope.entityList, start=1):
220                     file.write(f"\tEntity {j}:\n")
221                     file.write(f"\tName: {entity.name},")
222                     if isinstance(entity, self.Variable):
223                         file.write(f"\tOffset: {entity.offset}\n")
224                     elif isinstance(entity, self.Function):
225                         file.write(f"\tFrame Length: {entity.framelength},")
226                         file.write(f"\tArguments:{entity.arguments}\n")
227 
228                     elif isinstance(entity, self.TemporaryVariable):
229                         file.write(f"\tOffset: {entity.offset}\n")
230                 file.write("\n")
```

Εικόνα 42

7)Τελικός κώδικας

Για τον πίνακα συμβόλων δημιουργήθηκε η κλάση Final (Εικόνες 43,44,5 & 46) που περιέχει της βοηθητικές συναρτήσεις των διαφανειών και του handbook (αντί να υπάρχει η produce υπάρχουν μέθοδοι για σχεδόν κάθε περίπτωση). Επίσης δημιουργήθηκε και η writeFunctionFinalCode(quadSubList) (παράγει τελικό κώδικα σύμφωνα με τις τετράδες που δέχεται) και readQuadList(quad) (βοηθητική για να δημιουργηθεί ο τελικός κώδικας) (Εικόνες 47,48,49,50,51 & 52) που βοηθούν στην συγγραφή του τελικού κώδικα. Επειδή δεν υπάρχει ο fp στον RISC-V χρησιμοποιήθηκε ο s0 στην θέση του .

```
class Final:
    def __init__(self, record_structure):
        self.record_structure = record_structure
        self.instructions = []

    def gnlvcode(self, var_name, reg):
        entity = self.record_structure.searchEntity(var_name)[0]
        entity_level = self.record_structure.searchEntity(var_name)[1]
        if entity is None:
            print (var_name,"has not been initialized")
            print("Running has failed")
            exit()
        current_level = len(self.record_structure.scopes)
        levels_up = current_level - entity_level
        if levels_up > 0:
            self.instructions.append(f"\tlw {reg}, -4(sp)")
        for _ in range(1, levels_up):
            self.instructions.append(f"\tlw {reg}, -4({reg})")
        self.instructions.append(f"\taddi {reg}, {reg}, {entity.offset}")

    def loadvr(self, v, reg):
        def increment_reg(reg):
            prefix = reg[0]
            num = int(reg[1:])
            return f"{prefix}{num + 1}"

        if str(v).isnumeric():
            self.instructions.append(f"\tli {reg}, {v}")
            return

        entity_level = self.record_structure.searchEntity(v)[1]
        entity = self.record_structure.searchEntity(v)[0]
        if  entity is None:
            print (v,"has not been initialized")
            print("Running has failed")
            exit()

        current_level = len(self.record_structure.scopes)

```

Εικόνα 43

- `__init__(self, record_structure)`: Αυτή η μέθοδος είναι ο constructor της κλάσης. Δέχεται ένα αντικείμενο `record_structure`, που πιθανόν να αναπαριστά τη δομή του πηγαίου κώδικα. Αρχικοποιεί τα πεδία `record_structure`, που αναφέρεται στη δομή του πηγαίου κώδικα, και `instructions`, που αποθηκεύει τις εντολές εκτέλεσης.

- `gnlvcode(self, var_name, reg)`: Αυτή η μέθοδος δημιουργεί κώδικα που επιστρέφει τη διεύθυνση μνήμης ενός ονόματος μεταβλητής. Χρησιμοποιείται κυρίως για τη διεύθυνση της μεταβλητής σε ένα εξωτερικό επίπεδο εμβέλειας.

- `loadvr(self, v, reg)`: Αυτή η μέθοδος δημιουργεί κώδικα που φορτώνει την τιμή μιας μεταβλητής ή μιας σταθεράς σε ένα ειδικό καταχωρητή (register) `reg`. Η increment_reg(reg) δημιουργήθηκε για να δίνει τον επόμενο καταχωρητή

- `storerv(self, v, reg)`: Αυτή η μέθοδος δημιουργεί κώδικα που αποθηκεύει την τιμή από ένα ειδικό καταχωρητή (register) `reg` σε μια μεταβλητή `v`.

```
        if entity_level == 1:
            self.instructions.append(f"\tlw {reg}, -{entity.offset}(gp)")
        elif entity_level == current_level:
            if isinstance(entity, RecordStructure.Variable) or isinstance(entity, RecordStructure.TemporaryVariable):
                self.instructions.append(f"\tlw {reg}, -{entity.offset}(sp)")
            elif isinstance(entity, RecordStructure.RecordArgument) and entity.parMode == "CV":
                self.instructions.append(f"\tlw {reg}, -{entity.offset}(sp)")
                self.instructions.append(f"\tlw {reg}, 0({reg})")
                next_reg = increment_reg(reg)
                self.instructions.append(f"\tlw {next_reg}, 0({reg})")
        elif entity_level < current_level:
            self.gnlvcode(v, reg)
            self.instructions.append(f"\tlw {reg}, 0({reg})")
            next_reg = increment_reg(reg)
            self.instructions.append(f"\tlw {next_reg}, 0({reg})")

    def storerv(self, v, reg):
        entity_level = self.record_structure.searchEntity(v)[1]
        entity = self.record_structure.searchEntity(v)[0]
        if  entity is None:
            print (v,"has not been initialized")
            print("Running has failed")
            exit()

        current_level = len(self.record_structure.scopes)
        if entity_level == current_level:
            self.instructions.append(f"\tsw {reg}, -{entity.offset}(sp)")
        elif entity_level < current_level:
            if isinstance(entity_level, RecordStructure.RecordArgument) and entity.parMode == "CV":
                self.gnlvcode(v, reg)
                self.instructions.append(f"\tsw {reg}, 0({reg})")
        elif entity_level == 1:
            self.instructions.append(f"\tsw {reg}, -{entity.offset}(gp)")
```

Εικόνα 44

- `end(self)`: Αυτή η μέθοδος δημιουργεί κώδικα που τερματίζει την εκτέλεση προγράμματος.

- `callFun(self, funName)`: Αυτή η μέθοδος δημιουργεί κώδικα που καλεί μια συνάρτηση με το όνομα `funName`.

- `retToCaller(self)`: Αυτή η μέθοδος δημιουργεί κώδικα που επιστρέφει στον καλούντα κώδικα από μια συνάρτηση.

- `jump(self, jumpName)`: Αυτή η μέθοδος δημιουργεί κώδικα που πραγματοποιεί άλμα (jump) σε μια ετικέτα με το όνομα `jumpName`.

```
72      def end(self):
73          self.instructions.append("\tli a0, 0")
74          self.instructions.append("\tli a7, 93")
75          self.instructions.append("\tecall")
76
77      def callFun(self, funName):
78          self.instructions.append(f"\tjal ra, {funName}")
79
80      def retToCaller(self):
81          self.instructions.append("\tlw ra,0(sp)")
82          self.instructions.append("\tjr ra")
83
84      def jump(self, jumpName):
85          self.instructions.append(f"\tj {jumpName}")
86
87      def label(self, labelName):
88          self.instructions.append(f"{labelName}:")
89
90      def move(self, r1, r2):
91          self.instructions.append(f"\tmv {r1}, {r2}")
92
93      def operations(self, r1, r2, r3, op):
94          if op == "+":
95              self.instructions.append(f"\tadd {r1}, {r2}, {r3}")
96          elif op == "-":
97              self.instructions.append(f"\tsub {r1}, {r2}, {r3}")
98          elif op == "*":
99              self.instructions.append(f"\tmul {r1}, {r2}, {r3}")
100         elif op == "/":
101             self.instructions.append(f"\tdiv {r1}, {r2}, {r3}")
102         elif op == "not":
103             self.instructions.append(f"\tnot {r1}, {r2}")
104         elif op == "or":
105             self.instructions.append(f"\tor {r1}, {r2}")
106         elif op == "and":
107             self.instructions.append(f"\tand {r1}, {r2}")
108
```

Εικόνα 45

- `label(self, labelName)`: Αυτή η μέθοδος δημιουργεί μια ετικέτα με το όνομα `labelName`.

- `move(self, r1, r2)`: Αυτή η μέθοδος δημιουργεί κώδικα που αντιγράφει την τιμή από έναν καταχωρητή (register) `r2` σε έναν άλλον καταχωρητή `r1`.

- `operations(self, r1, r2, r3, op)`: Αυτή η μέθοδος δημιουργεί κώδικα για διάφορες πράξεις (π.χ. πρόσθεση, αφαίρεση) μεταξύ των τιμών που βρίσκονται σε τρεις καταχωρητές (`r1`, `r2`, `r3`).

```
108
109     def branch(self, r1, r2, label, con):
110         if con == "==":
111             self.instructions.append(f"\tbeq {r1}, {r2}, {label}")
112         elif con == "!=":
113             self.instructions.append(f"\tbne {r1}, {r2}, {label}")
114         elif con == ">=":
115             self.instructions.append(f"\tblt {r1}, {r2}, {label}")
116         elif con == "<":
117             self.instructions.append(f"\tbge {r1}, {r2}, {label}")
118
119     def write_instructions(self,s):
120         for instr in self.instructions:
121             s+=(instr + "\n")
122         self.instructions = []
123         return s
124
```

Εικόνα 46

- `branch(self, r1, r2, label, con)`: Αυτή η μέθοδος δημιουργεί κώδικα που υλοποιεί προγραμματικά την εκτέλεση προγράμματος σε περίπτωση που μια συγκεκριμένη συνθήκη είναι αληθής ή ψευδής.

- `write_instructions(self,s)`: Αυτή η μέθοδος δημιουργεί μια συμβολοσειρά που περιέχει τις εντολές εκτέλεσης που έχουν δημιουργηθεί και είναι αποθηκευμένες στο πεδίο `instructions`

```
1325        def writeFunctionFinalCode(quadSubList):
1326            global finalCode
1327            global framelen
1328            branchRelOperators = ["=",">","<","!=",">=","<="]
1329            arithmeticOperators = ["+","-","*","/","and","not","or"]
1330            index = 0
1331            while(index<len(quadSubList)):
1332                quad = quadSubList[index]
1333                currentLabel = getNextLabel()
1334                finalCode += currentLabel+":\n"
1335                if(quad[0] == "begin_block" and quad[1] == "main"):
1336                    finalCode += "Lmain:\n"
1337                
1338
1339                if(quad[0] == "begin_block"):
1340                    funcBeginLabels[quad[1]] = currentLabel
1341                    finalCode +="\tsw ra,0(sp)\n"
1342
1343                elif(quad[0] == "end_block"):
1344                    final.retToCaller()
1345                    finalCode = final.write_instructions(finalCode)
1346
1347                elif(quad[0] == ":="):
1348                    if(recordStructure.searchEntity(quad[1]) is not None):
1349                        final.loadvr(str(quad[1]),"t0")
1350                        finalCode = final.write_instructions(finalCode)
1351                    else:
1352                        final.gnlvcode(str(quad[1]),"t0")
1353                        finalCode = final.write_instructions(finalCode)
1354
1355                    final.storerv((quad[3]),"t0")
1356                    finalCode = final.write_instructions(finalCode)
1357
1358                elif(quad[0] in branchRelOperators):
1359                    final.loadvr(str(quad[1]), "t0")
1360                    finalCode = final.write_instructions(finalCode)
1361                    final.loadvr(str(quad[2]), "t1")
1362                    finalCode = final.write_instructions(finalCode)
1363                    final.branch("t0","t1","L"+str(quad[3]),str(quad[0]))
1364                    finalCode = final.write_instructions(finalCode)
1365
```

Εικόνα 47

```
1366            elif(quad[0] == "jump"):
1367                finalCode +=("\tj L"+str(quad[3])+"\n")
1368
1369            elif(quad[0] == "ret"):
1370                finalCode += "\tlw t0 -8(sp)\n\tlw t1, -"+str(pos)+"(sp)\n\tsw t1, 0(t0)\n"
1371
1372            elif(quad[0] in arithmeticOperators):
1373                if(str(quad[1]).isnumeric()):
1374                    if(quad[0] == "+" or quad[0] == "-"):
1375                        finalCode += "\taddi t0, zero, "+str(quad[1])+"\n"
1376                    elif(quad[0] == "*" or quad[0] == "/"):
1377                        finalCode += "\muli t0, "+str(quad[1])+"\n"
1378                else:
1379                    final.loadvr(str(quad[1]),"t0")
1380                    finalCode = final.write_instructions(finalCode)
1381
1382                if(str(quad[2]).isnumeric()):
1383                    if(quad[0] == "+" or quad[0] == "-"):
1384                        finalCode += "\taddi t1, zero, "+str(quad[2])+"\n"
1385                    elif(quad[0] == "*" or quad[0] == "/"):
1386                        finalCode += "\muli t1, "+str(quad[2])+"\n"
1387                else:
1388                    final.loadvr(str(quad[2]),"t1")
1389                    finalCode = final.write_instructions(finalCode)
1390
1391                final.operations("t0","t1","t2",str(quad[0]))
1392                finalCode = final.write_instructions(finalCode)
1393                final.storerv(str(quad[3]),"t2")
1394                finalCode = final.write_instructions(finalCode)
1395
```

Εικόνα 48

```
1396            elif(quad[0] == "mod"):
1397                if(recordStructure.searchEntity(quad[1]) is not None):
1398                    final.loadvr(str(quad[1]),"t0")
1399                    finalCode = final.write_instructions(finalCode)
1400                else:
1401                    final.gnlvcode(str(quad[1]),"t0")
1402                    finalCode = final.write_instructions(finalCode)
1403
1404                if(recordStructure.searchEntity(quad[2]) is not None):
1405                    final.loadvr(str(quad[2]),"t1")
1406                    finalCode = final.write_instructions(finalCode)
1407                else:
1408                    final.gnlvcode(str(quad[2]),"t1")
1409                    finalCode = final.write_instructions(finalCode)
1410
1411                final.loadvr(str(quad[3]),"t2")
1412                finalCode = final.write_instructions(finalCode)
1413                finalCode+= "\trem t2,t0,t1\n"
1414                final.storerv(str(quad[3]),"t2")
1415                finalCode = final.write_instructions(finalCode)
1416
1417            elif(quad[0] == "inp"):
```

Εικόνα 49

```
1413
1414            elif(quad[0] == "inp"):
1415                final.loadvr(str(quad[1]),"t0")
1416                finalCode = final.write_instructions(finalCode)
1417                finalCode+= "\tli a0, 0\n\tli a2, 1\n\tli a7,63\n\tecall\n"
1418                final.move("t0","a0")
1419                finalCode = final.write_instructions(finalCode)
1420
1421            elif(quad[0] == "out"):
1422                final.loadvr(str(quad[1]),"t0")
1423                finalCode = final.write_instructions(finalCode)
1424                finalCode+= "\tlw a0, 0(t0)\n\tli a7,1\n\tecall\n"
1425
1426            elif(quad[0] == "halt"):
1427                final.end()
1428                finalCode = final.write_instructions(finalCode)
1429
1430            elif(quad[0] == "par"):
1431                finalCode+= "\taddi s0,sp,"+str(framelen)+"\n"#### opou fp s0
1432                while(quad[0] =="par" and quad[2]!="ret"):
1433                    entity,entity_level = recordStructure.searchEntity(quad[1])
1434                    offset = entity.offset
1435                    finalCode += getNextLabel()+":\n"
1436                    final.loadvr(str(quad[1]),"t0")
1437                    finalCode = final.write_instructions(finalCode)
1438                    finalCode+= "\tsw t0, -"+str(offset)+"(s0)\n"###opou fp s0
1439                    index+=1
1440                    quad = quadSubList[index]
1441
1442
1443                finalCode += getNextLabel()+":\n"
1444                finalCode+= "\taddi t0, sp, -"+str(offset)+"\n"
1445                finalCode+= "\tsw t0, -8(s0)\n"###opou fp s0
1446                index+=1
1447                quad = quadSubList[index]
1448                finalCode += getNextLabel()+":\n"
1449                label = funcBeginLabels[quad[1]]
1450                final.callFun(label)
1451                finalCode = final.write_instructions(finalCode)
1452            index+=1
```

Εικόνα 52

```
1454
1455    def readQuadList(quad):
1456        funcName = quad[1]
1457        endIndex = quadList.index(quad)
1458        beginIndex = quadList.index(["begin_block",funcName,"_","_"])
1459        writeFunctionFinalCode(quadList[beginIndex:endIndex+1])
1460
```

Εικόνα 52